De-Layered Grid Storage Server

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

Abstract

Networks have become faster and disks have become fatter at a pace that, despite Moore's law, CPU developments have simply not been been able to keep up with. We present a Grid Storage Server which is capable of scaling up to meet the "terabit-terabyte" demands of very large scale grid computation applications with large data sets.

The Grid Storage Server is implemented almost completely in silicon, whether FPGA or ASIC; the fast-path of this server does not use a CPU or von Neumann style (instruction/data) machine. Instead, multiple layers of a protocol stack are compiled into a hardware engine that processes all layers concurrently on-chip. This "serverless" design allows it to scale up to match "terabit" network speeds and "terabyte" disk capacities enabling large scale grid applications.

At a price-point a small percent of that of a server-based design, the Grid Server incorporates a standards compliant high-performance TCP stack that can saturate 40Gbps using a single or multiple TCP connections. The current design directly attaches to a storage array of 48TB capacity. The storage array is organized with a fault-tolerant RAID for performance and reliability; the RAID configuration is adaptive and can be tuned to conflicting application needs. As a bonus, because the control-plane in the silicon-based TCP engine has very low jitter, the protocol engine has mechanisms for nanosecond precision clock synchronisation across very large distances, thus, for the first time, enabling transcontinental real-time and temporal grid computation and database applications.

1 Introduction

The following two trends in computing are well known – First, where demand for computational horse-power has outstripped Moore's law, the notion of cluster computing has provided an answer, by harnessing aggregate CPU power across many machines[19]. Second, wide-area network links with bitrates approaching 40Gbps per lambda are now being deployed; these speeds are comparable to CPU-memory bitrates within traditional computer systems. This two trends have together enabled compute clusters to be geographically dispersed, which is the notion of Grid Computing.

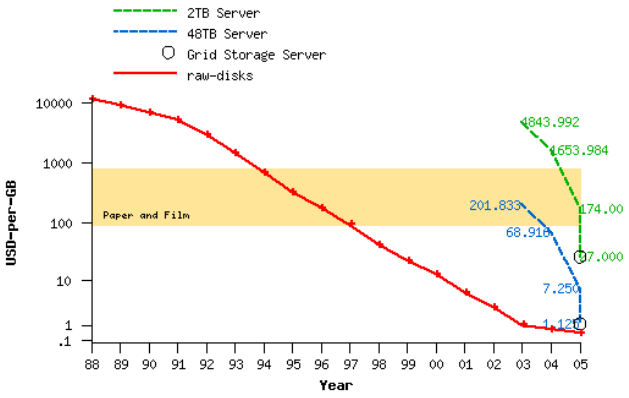

Figure 1: Storage and Server Cost Trends

Such architectures are applicable to "Big Science" applications[19,22] as well as data-hungry distributed commercial applications such as Decision Support[23].

The comparable trends in storage are more interesting. While server and disk-drive unit costs have remained almost stationary, disk density has been growing at 100% CAGR[20] which is significant because it is even faster than Moore's Law (25-43% CAGR). Thus disks with areal densities of $100\ Gb/in^2$ disks are around the corner[21]. At the same time disk access-times and access-rates have not kept up. Together this means future applications would of necessity demand clusters of disks aggregated into virtual file-systems.

Typically, clustered storage-servers are deployed in a Computation Grid to (1) organize the raw bulk data storage into usable file-systems, (2) to perform locking and other consistency maintaining and co-ordinating procedures across the multiple geographically scattered users, and (3) to serve the data on protocols that are compatible with wide area networks, for eg. storage protocols that are layered over the TCP/IP suite. With currently available implementations, these protocols require software implementations[2] which therefore continue to be hosted on servers of traditional von Neumann architecture. TCP-offload engines (ToEs)[1,25] are a partial alternative to software TCP stacks; while they help to augument server capacity, they themselves continue to be limited by scaling with Moore's law, which we have observed is not fast enough.

This can be seen in Figure 1 which compares the cost of digital storage against traditional paper and film. It can be

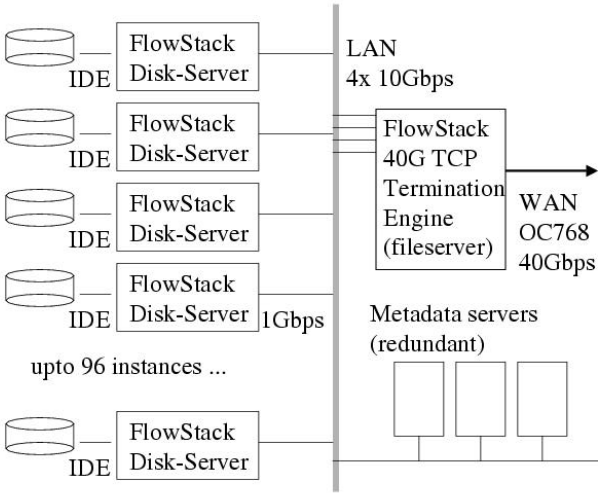

Figure 2: ”Serverless” Grid Storage Server

seen that with the introduction of small form-factor disks in 1996, the per-megabyte cost of raw digital storage (red line) has fallen below that of paper. However, the cost of a fully loaded server with disks continues to be much higher. A few example price points for a 2TB server, a typical ”Enterprise” configuration (green-line), show a cost range of 4800 to 175 USD-per-GB (details in Section 5). It can be seen that with traditional server architectures, the cost of digital storage is not competitive yet against a 5000 year old medium.

In the work reported in this paper, we present Grid Storage Server, a distributed disk block- and file-server architecture well suited for Grid Computing applications that can be implemented at a significantly lower cost (27 USD-per-GB for 2TB, and approaching raw-disk costs for 48TB).

1.1 All-Silicon Grid Storage Server

The Grid Storage Server (Figure 2) is implemented using two different types of protocol engines, both based on the FlowStack machine described in this paper. It is backed by an array of 96 inexpensive disks that are directly attached to the IP networking infrastructure without any intervening software servers with traditional von Neumann CPUs. The FlowStack based disk-network-interface chips are inexpensive enough to be absorbed into the price of a commodity harddisk. The Grid Server can be realized in under $57,500 for such a configuration; implementations using conventional server-based technology which can be as high as $9.5mn for a comparable 48TB system.

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.

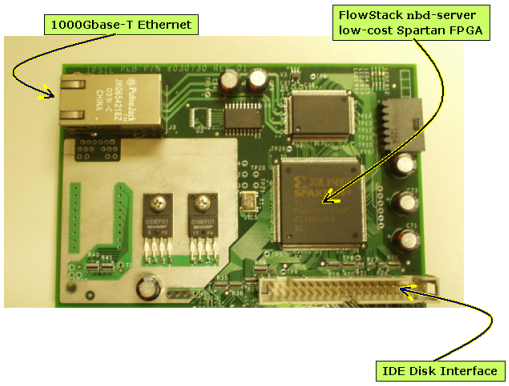

Figure 3: FlowStack disk-block server with IDE

1.1.1 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)[1]. It is implemented completely in silicon, on a single FPGA board with two 40Gbps interfaces (not shown).

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 5).

1.1.2 Disk Block Server: `nbd`

Figure 3 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.

1.1.3 High Precision and Low-jitter NTP

In addition, the TCP stack, being implemented in silicon, has very low and predictable jitter; of the order of the baud time of the interconnect on the ingress channel. With OC-768 SONET interfaces, this jitter can be globally bounded

[1] Perhaps it is currently the world’s fastest TCP stack

in the nanosecond range. With care, this can be used to augument global time-synchronization protocols, such as NTP, to improve their precision by 5 or 6 orders of magnitude compared to a software NTP implementation. Such a precise NTP service can be used by temporal and real-time databases and transaction managers implemented on the Grid Server. This nsec-precision time-synchronisation mechanism also enables several trans-continental grid-based distributed scientific computation applications, eVLBI radio astronomy (extremely-Very Large Baseline Interferometry) being one example.

2 Grid Workload Characteristics

As grid applications evolve, the next-generation of widely deployed grid applications are expected to work with datasets several orders of magnitude larger than anything so far encountered. The Large Synoptic Survey Telescope (LSST) uses a 3-gigapixel camera will produce up to 20 terabytes of data per night. Weather models commissioned by the Intergovernmental Panel on Climate Change (IPCC) produce 7.5TB of data for each 100-year simulation. The Large Hadron Collider (LHC) will produce close to a petabyte per second of raw observations, which will be culled down to 1500 GB per day[22]. In the commercial world[23], there are decision support databases with 29.2 TB, and transaction system databases with 18.3 TB.

As a specific motivating example, we use the newly evolving radio-astronomy technique of eVLBI (extremely-Very Large Baseline Interferometry) which utilizes multiple radio-telescopes in different parts of the globe to simultaneously collect signals from the same part of the sky. In one recent proof-of-concept experiment[5,18], real-time fringes were detected using the EVN telescopes in Cambridge (UK), Torun (Poland) and Westerbork (Netherlands) and the 305m Arecibo dish observing ICRF reference source 0528+134. Ideally, a typical one hour observation would produce a broadband signal sampled at 800Msps with a 32bit resolution, collecting 5.7TB of data per hour per telescope. Raw data accumulated over several days will be repeatedly accessed as input to different kinds of analysis and experiments. Additionally, since the application requires time-synchronized measurement, the silicon supported high-precision NTP in FlowStack could also be very useful.

3 Protocol Engine Component

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 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.

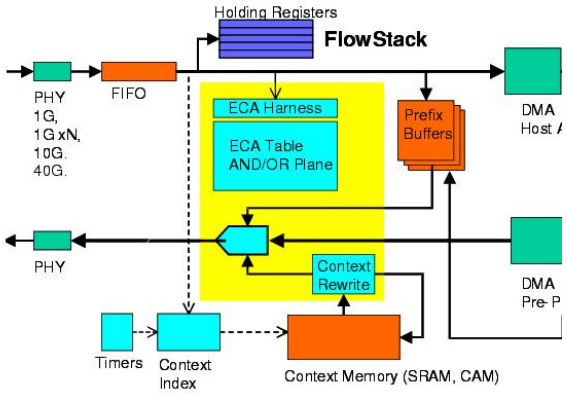

Figure 4: FlowStack machine architecture

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 Architecture

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

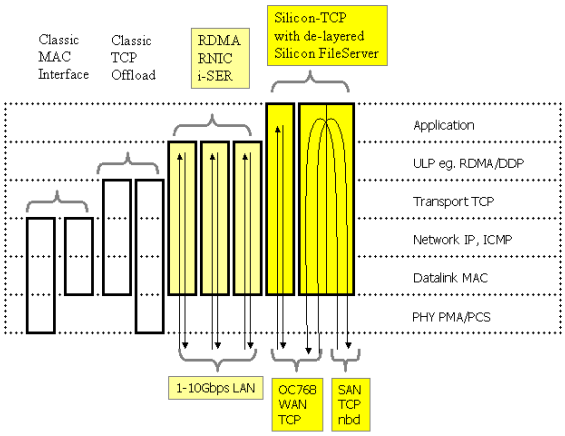

Figure 5: Slices - De-layered Programming Model

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 down-count, 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[2] 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, the steps that lead to its implementation are modular, with striking resemblence to software development.

Each such hand-optimzed 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 that may have been inadvertently coded. Any conflicts that are detected are resolved as follows:

1. Protocol Functional Verification stage: A conflicting case is a packet that is claimed by two or more ECA slices. The semantics is formally disambiguated by specifying a unique priority order during compositing of the slices. This resolution of a conflicting case can be analyzed for formal correctness using reachability and bisimulation[16] against a traditional software implementation. Thus, the formal equivalence and hence correctness of this silicon implementation can be compared against a known-good, albeit slower, traditional software reference implementation.

2. Compositing Time: The ECA slice compositing tool can identify and flag all the overlap cases. These overlap cases can be verified against the list of overlaps that have been explicitly analyzed during the protocol functional verification stage. Any errors introduced after the Protocol Verification Stage will thus be flagged during automated compositing.

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. Barring bugs in the compositing tool, such cases are not possible; this checking stage thus acts as a security against implementation errors in the Compositing Tool.

4. At Runtime: The remaining logical conflicts between cases in the ECA table slices have now been separated

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

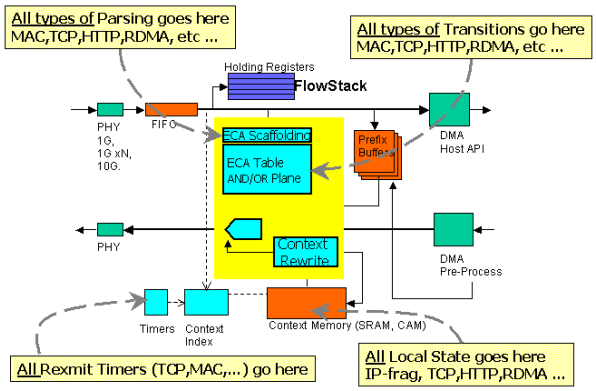

Figure 6: De-layering and Compositing

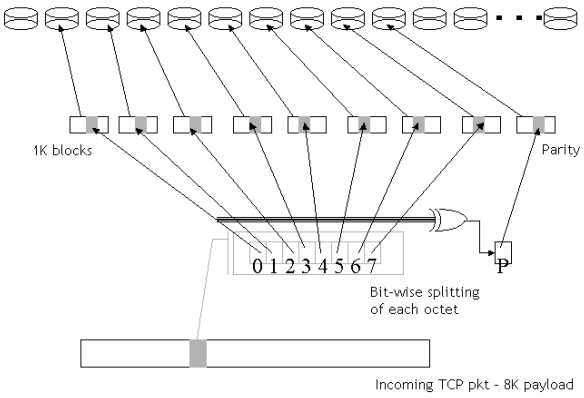

Figure 7: Bitwise RAID - write operation

by explicit prioritization of the choices. Each such conflict case gets reduced into priority encoder by the synthesis tool in the final ECA table logic. Again, functional correctness is assured. The slight speed reduction contributed by each such priority encoder is an inherent cost in that protocol stack; this cost can be mitigated by redesigning the protocols themselves or by tuning their constants to avoid overlap cases. Again, searching in the critical path of the synthesized logic for priority encoder structures will give a strong hint about the higher level protocol elements whose conflict contributes to the slow-down. Having obtained such a hint from the synthesized logic, it is easy to reflect the knowledge back to the high level ECA slices and re-iterate the design. Unlike in the case of hand-crafted custom ASIC implementation of protocol engines, it is not necessary to hand-craft the logic solution to the speed bottleneck at the low-level of random logic.

Upon compositing, all the slices of the ECA table are reduced to form a single large register-free plane of boolean logic. This structure is itself not intended to be manually edited. Instead it is passed on to synthesis tools (silicon compilers) which then reduce the ECA table into actual gates in the final target silicon device.

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 6). 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 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.

4 File System Component

4.1 RAID5 Operation

Unlike a software stack where generally entire packets are assembled into linear buffers before processing, a silicon protocol engine allows (and generally requires) all processing to be be done deterministically, in a fixed number of clock cycles and at wire-speed. This means that, in the FlowStack engine, it is almost as inexpensive to do byte-level (or even bit-level) manipulations on files as it is to do block level operations; there is no large fixed-cost of an invocation, context-switch

or interrupt-request that has to be amortised across the number of bytes, and thus requring a large buffer.

Secondly, due to the de-layering enabled by the programming of the FlowStack ECA core in units of vertical slices, knowledge about the fate of the packet at as high as the application layer is available even to the physical and network layer behaviours. In other words, this is a form of interlayer processing[26] in which it is possible to determine the disposition fate of practically every octet in a packet almost at wire-time, as they arrive.

Together, the above two features enable a unique form of RAID5 implementation. Let us consider that, as a packet arrives, at wire-time, its headers are parsed and it is determined that the packet is carrying a data write to a file. As shown in Figure 7, each octet is split into 8-bits, and a parity bit is computed. These 9-bits are then accumulated into buffers which are written into bit-wise and read out byte-wise. Thus a single 8-Kbyte write PDU from the network is broken into nine 1-Kbyte data blocks which are then sent to the respective disk drives. The parity scheme can be generalized to any number of bits or be replaced by a more efficient ECC mechanism.

A unique advantage of this RAID-5 is that it does not suffer from the short-write problem[8,9] that plagues all software RAID implementations; from a storage network viewpoint of computing the parity, writing a single octet is almost as inexpensive as writing an entire disk block. Writes shorter than 512 bytes sectors are implemented as a read-modify-write operation by the FlowStack core in the disk block servers, but this is a fast operation as it operates out of on-disk buffers and over a dedicated IDE/SATA interface with spare capacity.

4.2 File System Metadata

The Grid Server uses a filesystem whose Fast Path is implemented in the FlowStack engines in silicon. The meta-data of the file-sytems, along with all policy and permission control, is not accelerated, but implemented in traditional software file-server machines.

A network PDU carrying a file-`open()` operation is parsed by the TCP-termination engine and the parameters are passed on to the meta-data servers. After appropriate access control, the meta-data servers communicate back to the TCP-termination engine a block-allocation control-block which identifies a (possibly pseudo-random) `algorithm` to map octets (file-pointer or equivalently TCP-sequence number) to blocks to disk-drives, and an `algoseed` which offsets the start within this block-allocation sequence. This information is saved along with the TCB (TCP control block) and allows the TCP-engine to perform all RAID and nbd computations autonomously, at wire-time.

The RAID controller embodied within the FlowStack engine can be programmed to support most known RAID mechanism[8,9,10,11,13,14,15], such as RAID5, RAID10 or ROWB. Also, this choice can be made independantly for each file in the data store, or even each generation of a file. A write to a file that advances its generation count can be concurrently supported while a read of the previous generation is still in progress by bifurcating the file into extents; operations on file-offsets within the two extents are governed by different `control-blocks` whose parameters have no collisions on physical disk-blocks.

5 Cost Comparisons

In the following section, we compare the price point achieved by the prototype Grid Storage Server against a traditional storage server built out of off-the-shelf components configured for a similar aggregate capacity and performance. Along each separate dimension, the Grid Storage Server produces cost savings of an order of magnitude or more.

5.1 RAID Controller Costs

In a traditional storage server, the RAID functionality is implemented using commercial RAID controllers for high performance. These RAID controllers add significantly to the cost of the system. As a point of comparison, a typical RAID controller bank for a traditional PCI-architecture server (Intel SRCU42L, each rated at 320MB/s thus needing 16 units to aggregate to 40Gbps), has a street-price of approximately $7787.

In the Grid Storage Server, the RAID functionality is implemented within the same FlowStack protocol engine, along with the TCP/IP Offload and the block server upper layer protocol, and therefore has zero incremental cost.

5.2 Server System Costs

In terms of storage costs, the Grid Storage Server approaches the ideal of `Data Bricks` proposed by Jim Grey[7]. The incremental cost of the silicon for the FlowStack protocol engine per network-attached disk is in the region of $25 (Xilinx Sparatan). The cost of the FPGA used to implement the TCP termination engine at the 40Gbps network attachment is in the $500 range. Adding the cost of the I/O transcivers ($3000 for the SFI-5 optics to connect to the OC-768 links, plus $4000 for 96 instances of gigabit copper PHYs), the total cost of the server subsystem is in the $9500 range. The cost of the bank of 96 inexpensive commodity IDE disks will add another $48,000 to the price, bringing the total system cost of the server subsystem to about $57,500 for a storage server.

In comparison, the recent demonstration of 100Gbps of aggregate TCP transmission at SC2004[17] used 10Gbps TCP offload engines costing approximately $2000 each, hosted on servers that cost in the order of $10,000 each. The cost of the server infrastructure comes up to approximately $188,000;

however even this pales in comparison to the cost of a fully configured disk storage array.

The raw storage costs dominate the application; a 48TB Fibre Channel or SCSI array (eg. NetApp FAS960c or Sun StorEdge 3310) would add $3.12 to 9.5 million. SATA based storage would be lower in cost; an EqualLogic PeerStore PS-series block server built out of SATA disk arrays would cost only about $685K for the same 48TB capacity. A 48TB BladeStore array from StorageTek, built out of ATA disks would only cost $160K. These are for raw storage, with no application and transport protocol termination.

6 Conclusions

The Grid Storage Server, built using the silicon-implemented protocol engine based on the FlowStack architecture demonstrates that the storage subsystem costs of a Computational Grid can be brought down by an order of magnitude or more. A server for a 48TB disk array can be configured for a price that is two orders of magnitude less than that of conventional server architectures.

In addition, the Grid Storage Server can saturate a 40Gbps pipe with a single or multiple TCP connection(s); the traditional server implementations can do so only using multiple parallel TCP connections. The silicon implementation of protocol engines in the Grid Storage Server is significantly more faster and deterministic. This allows FlowStack augmented NTP to provide nanosec time-synchronisaton.

The Grid Server described in this paper is currently being assembled in prototype form. The components that make up the system have been succesfully implemented and quantified either on a lab bench or in simulation.

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, Oct 2001, pp74-79, Volume 39, Issue 10

[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, pages 456–465, 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, Pennsylvania, USA, pp.279-286.

[16] R. Milner, J. Parrow, D. Walker : "A Calculus of Mobile Processes - Part I" – LFCS Report 89-85. University of Edinburgh June 1989.

[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, MarchMay 2004

[24] Winter Consulting's 2003 survey of Largest DBs, 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.