The All-IP Manifesto

Rodney Van Meter
September 17, 2009

Abstract

A few thoughts (brain storming? free association? dignifying this document with a name is a little much...) on the technical direction and desirable background reading for the All-IP project. Based on current technological trends, I believe that the boundaries of systems are evaporating, and that heterogeneity in latency is increasing, leading to the buzzwords boundary-less systems *and* latency-tolerant computation*. I believe we need to speculate more during computation.*

1. Introduction

All-IP Computing starts from a very simple premise: *the Internet Protocol (IP) is the only communications mechanism you need.* From this idea flows research in three areas:

- *Component Interconnects:* iSCSI has already proved to be a commercial success, replacing the physical connection to a disk or tape with SCSI commands sent over a network. We work on advanced storage device protocols, USB/IP, IP-based display protocols, and boldly propose that IP can even be used in place of a memory bus. One of the most important issues in this area is the *operational semantics* for devices. Do they need to change when latency or sharing increases?
- *Planetary-Scale Dynamic Systems:* Also called "cloud computing" or "utility computing", we are working toward on-demand allocation of computational and storage resources from around the globe.
- *Human-Centric Computing:* The most profound consequence of the All-IP philosophy is that *human beings* should be more evident in the computational model, supporting anytime, anywhere, virtually unlimited computing and access to *all* of one's data.

2. The Changing Landscape

Life in the late Moore's Law period:

- *multicore (and other parallel):* processor-to-processor latency is low, processor-to-memory latency is high, and the boundary of the system gets fuzzier.
- *virtualization:* everything except large physical racks of equipment can migrate.
- *search and data-centric computing:* managing your data is harder than creating it; Google wins, the desktop (and Microsoft?) lose.
- *asynchronous:* AJAX.
- *mobile and ubiquitous:* everything, everywhere, always on...

3. Problem Portrait

So, the big picture is that in light of the above changing landscape, it's getting harder to create systems that solve problems faster. It seems we are in danger of hitting some sort of serious "scalability wall", though the biggest systems (BlueGene, etc.) seem to do pretty well. The gap between what a wizard can achieve and what a normal programmer can achieve is large, though (and growing?).

It's also *still* hard to manage disparate computing devices and resources safely and with minimal effort.

...but those are both too vague. We need a more clearly defined problem. Homework for all!

4. The Eternal Landscape

Readings you need to look into. Number one on the list is Lampson [53].

4.1 Distributed Operating Systems

Read the book, it's good [19]. I don't have a Japanese copy, but I believe it has been translated. Tanenbaum also has a good one, which I had a Japanese copy of...I think I lent it out...

Ah, and you should look at chapters 3-5 of Roy Fielding's Ph.D. thesis [27]. It will help you understand distributed information transfers. Likewise, you should go back

to the beginning on remote procedure calls (RPCs), which have been around since 1976 or earlier [11].

You really need to know about Sprite [65], V [16] (I always thought the triangle-routed RPCs in VMTP were clever [17]), Condor [58], VMS VAXclusters [51] (which was, arguably, the most commercially successful system with many "modern" distributed system ideas in it), Amoeba [78], Locus [87], and Plan 9 [67].

Process migration is one important topic; many papers exist on the topic, and I'm not an expert, but you should read one or two of the references [36, 63, 75, 23].

The above are operating systems; distributed and parallel programming models are also important.

4.2 Cloud/Utility and Parallel/Distributed Computing

Parallel computing, of course, is an enormous field, but you need to have some basic idea of how it works in message passing and shared memory forms.

Other old things that need to be understood:

1. I'm a big fan of Linda [14]. In Linda, information producers insert *tuples* into a shared, distributed tuple space, and consumers extract those tuples. All of the parallelism and distribution is then handled by the system. Scalability and efficiency are important and tricky. In some ways, it's like a simpler MapReduce (see below).
2. Virtual Time is a powerful insight into the way information must propagate in distributed systems [42]. It is based on a Newtonian concept of time, and was one of the things that prompted me to think of relativistic time as an organizing principle for distributed systems.
3. In that same vein, Joe Touch is found of saying, "Everybody complains about the speed of light, but nobody ever does anything about it," which is funny and motivational, but not quite true (arguably, all caching work is exactly that – an attempt to get around latency). Nevertheless, his thesis is an interesting attempt to bring some physical principles (including relativistic information transfer) to networking; I like it [81].
4. Joe also has a patent [80] for a "code pump", where the code pump is located near the memory and prefetches multiple instruction streams, with a filter placed closer to the processor.
5. Leases [34].

Some current hot buzzwords worth investigating are:

1. MapReduce [20]. Cited by many as a good, new alternative to message passing systems, which are often a hassle to program, maintain, administer, and optimize.
2. BOINC, the Berkeley Open Infrastructure for Network Computing [6], is the premiere *volunteer computing* system.
3. Utility [40], cloud, or service computing [40]. The publish/find/bind model is important. "Cloud computing" is such a new term that it shows up in many recent web articles, but few technical papers. However, all of these are related, and all related to many years of distributed computing research.
4. Eddies are another concept for dynamic allocation of transactional processing resources [9].
5. *Thread-level speculation* is an active area of research [18].

4.3 Distributed Storage

The first two places to start are the CACM paper by Garth and yours truly [33], and Levy and Silberschatz on distributed file systems [56]. I also recommend Riedel's customer dissatisfaction paper to understand how to *justify* performing a large chunk of work [70]. Two other useful papers here are Katz [43] and Sachs [72], especially describing how networking concepts gradually appear in I/O interfaces.

The 1980s and 1990s saw a bunch of things tried. When you read these in detail, you'll be surprised at how many of your current ideas are present there. That doesn't mean they were bad ideas, nor does it mean that there is no reason to try them again! Knowing why a project failed to become mainstream, what is different between the present and the time it was tried before, and how to extend prior ideas, are imperative. The things you need to know about, in order to carry on an intelligent conversation, listed in rough order of priority for reading, are:

1. Network Attached Secure Disk (NASD) [32], CMU. Object-oriented interface for the devices. Sophisticated firmware inside the device makes space allocation decisions.
2. Derived Virtual Devices (DVDs) [86], USC/ISI. Anyone with the right to use a device can create a new right for someone else to use a subset of that in a secure fashion, in the form of a derived virtual device. DVDs are not persistent, they are lost when power goes out.
3. Petal [55] and Frangipani [79], DEC. Frangipani includes a VMS-like distributed lock manager, with leases. Petal is the networked, snapshotted, block-level, virtual disk system Frangipani is built on.
4. OceanStore [52], Berkeley. Excellent ideas on using untrusted servers, though the implementation never really seemed to get off the ground.

5. xFS (serverless file systems) [7], Berkeley. Good FS work for a cluster; replicated and logged.
6. GoogleFS [31]. 'Nuff said.
7. Andrew AFS and Coda [74]. The granddaddies of wide-area file systems. If you are interested in dynamic, ad hoc caching in e.g. your current location, look at WayStations, too [45].
8. TickerTAIP [13], HP. A first step toward distributed RAID. A lower-priority read.
9. Swift/RAID [60], Sandy Eggo. Another early distributed RAID paper; also lowish-priority at the moment.
10. Zebra [37], Berkeley. Another approach to distributed, striped storage.
11. GFS [76], Minnesota. Direct use of shared disk drives at the block level, with a lock server.
12. RAID-II [25], Berkeley. Building a file server that uses a better control/data separation.
13. GridFTP [5], the Globus Project.
14. Sarkar *et al.* investigated when it is favorable to use a hardware implementation of iSCSI and when it isn't [73].
15. Pick one paper on out-of-core distributed I/O, and read it. Selections are available in the IOPADS book [41]. Alok Choudhary has been working this theme for more than a decade, and his group has an interesting paper on inter-file access patterns [61].

You should also read almost everything ever written by John Wilkes and Greg Ganger, and a smattering of work by Ethan Miller, Darrell Long, Randall Burns, and a few others.

4.4 File Systems

Eventually, I'll add a basic reading list on file systems here. Elephant, Plan9, log-structured, journaled, write-once, B-trees, WAFL...

Need RAID, too.

4.5 Distributed Hardware Architectures

You should know all of the buzzwords – NUMA, CC-NUMA, NORMA, RDMA, etc...of particular relevance to my vision going forward is cache-only memory architectures [35] like the KSR-1 AllCache [71].

Memory shared over a network goes back quite a ways [21, 77]; as long ago as 1996, it was possible to list 411 references on the topic [26], some for pure software implementations for parallel computation, others for hardware.

When discussing memory semantics, I think we need to consider *transactional memory* [54], which can be done in either hardware or software, and is touted as one solution to the difficulty of parallel programming.

In the multimedia area, you must know about Netstation [29], ViewStation [57, 3], and Desk Area Network [10].

Overall, there are a ton of references on multimedia and storage network-attached peripherals in my survey paper [84].

For reasons that become clear below, you should be familiar with the basic idea of *dataflow* architectures, which you can learn about from papers [22] or textbooks.

AMD supports HyperTransport and provides a cache coherent, non-uniform memory access (CC-NUMA) architecture.

4.6 Multi-Core Chips and Networks

Absolutely critical to understand, not just for All-IP but in general. Also called *chip multiprocessors* and *many-core chips*. You should read about the Sony/Toshiba/IBM Cell Processor, Sun's Niagara, and Intel's 80-core chip.

For network-on-chip (NoC) architectures, check out Bjerregaard and Mahadevan [12], as well as the special section in the Sept. 2008 issue of *IEEE Transactions on Computers*.

4.7 Virtualization

As I said at the top, one of the most popular concepts in current systems work, and a key to enabling migration of services, in my opinion. We used to virtualize processes; now we virtualize entire machines and move them around, a la VMotion. Read about Xen [24], and maybe go back and read the original formalization of the topic [68]. There was a special section in an issue of *IEEE Computer* (May 2005) on the topic [28]. Some of the papers are very good; you should pick one or two and read them.

4.8 Fault Tolerance

A huge area in which we need at least a smattering of help. Who here has taken a class in FT?

Random suggestions: practical Byzantine fault tolerance has gotten a lot of mileage [15].

4.9 Parallel Languages and Compilers

A list is needed here...we're not going to attack this problem directly, but we need to know about it...

Random references: [59, 4]

4.10 Resource- and Environment-Adaptive Computing

Odyssey [62], as well as lcron [38], Netstation [29], etc.

5. Buzzwords of Choice

5.1 Boundary-less Systems

With the advent of many-core chips, we will have processors that have much lower latency to *each other* than to "local" memory.

iSCSI (and, to a lesser extent, Fibre Channel) erases the boundary for device I/O. It used to be that a device was under the complete control of one instance of a host operating system; today that it no longer true.

Interestingly, virtualization, which is becoming increasingly critical in data center environments, also serves to fuzz system boundaries. A VMware/VMotion-enabled system, when it moves, continues to access devices on its original "home" hardware platform.

5.2 Latency-Tolerant Computation

When multiple many-core chips are built into a system, latency to memory and processor-to-processor latencies differ, and the cache consistency protocols become more difficult.

Optimizing computation becomes more difficult when access to different memories varies, and especially when access to the *same* memory or device varies over time – as when a VMware server instance migrates to other hardware.

In early distributed systems design, it was usually a *process* that was migrated to balance load or provide fault tolerance. Now, it is often a *virtual machine*.

One aspect of latency tolerance is flexibility in the ordering of computations; I proposed a file system interface called SLEDs, storage latency estimation descriptors, that allows an application to do the easy work first and postpone more time-consuming operations [85].

SLEDs allow a form of "dataflow" computation at a macroscopic level; one idea I proposed long ago (but never put on paper in any form other than the SLEDs work) is a *dataflow operating system*. I still think the idea is fascinating and valid, and works well in the kinds of heterogeneous systems that are being built. However, in its most obvious form, it requires re-architecting computationally complex applications, and is a very different computational model. The more object-oriented your system is, probably the cleaner it fits into this kind of model.

6. Pick Your Battles

Here are a series of problems, each of which should be worth at least one paper, and some of which are worth a Ph.D. thesis (or two, or three).

6.1 Memory Interconnect

Macchan-san's work has already taken a big stride toward showing that IP can be used as a memory interconnect. A little more experimentation, and writing up a paper, and we're all rich, famous, and covered in glory for solving one of Mankind's biggest problems.

⇒ Demonstrate that IP can be used as a memory interconnect.

6.2 Memory Semantics

In systems with heterogeneous latency, is transactional memory useful [54]? Should devices perform a form of data filtering, as the IRAM project proposed more than a decade ago [48]?

⇒ Variable-latency transactional memory.

⇒ How about Linda memory?

6.3 Scalability in Latency-Heterogeneous Caches

Hennessy & Patterson has an excellent section on hardware protocols for cache consistency [39], and it was translated into Japanese by Keio's own Hunga-sensei. Understanding the basic cache protocols and the performance impact of cache misses and sharing conflicts is critical; the Linux kernel gives a process an affinity for a single processor for exactly this reason – moving a process from one process to another, or creating a single, shared list of runnable processes that all processors pick from arbitrarily, results in cache thrashing poor performance.

But existing cache protocols generally assume a fairly uniform physical environment, for reliability, security, latency, and processing speed. What happens when those assumptions break down?

⇒ Could we, for example, create a *multi-granularity sharing* scheme that would scale more smoothly? (How would this differ from a multi-level cache?)

6.4 Device Semantics

In addition to the NASD/OBD, DVDs, and other things mentioned above, some systems proposed a more programmable access to the device. We did a project like that at Quantum which was an utter failure; research projects in what are called "Active Disks" came from CMU [69], Santa Barbara [83], and again from Berkeley [44]. (Those three papers are similar enough that it's only necessary to read one.)

⇒ Active disks might be a good platform for MapReduce, as Heidemann has pointed out.

What I want to know about the interface to a node/device on the network is:

1. what is the command interface?
   (a) blocks, bytes or pixels
   (b) files, or windows
2. how much does it understand about the security (authentication/authorization) scheme?
   (a) nothing (must be protected; will process any command it receives from anyone)
   (b) can enforce a specific policy (e.g., only execute commands signed with the proper key), but doesn't understand anything about "users"
   (c) can participate in full security, understands UIDs, permissions, etc.
3. can it do third-party transfer?
   (a) A tells B, "Send block #123 to node C, address 0xDEADBEEF."
   (b) How are errors reported?
   (c) Can data formats be massaged before sending? (managing mismatched block sizes, move disk blocks to screen rectangles, etc.)
4. can it handle concurrency control, or does a manager (or managers) have to control access to the device?
5. how do we do device discovery?

6.5 Resource Discovery and Macroscopic Sharing

Resource discovery, in general, is a problem that has remained unsolved. LDAP and various Microsoft-specific things work on a LAN, or with generous assumptions about security. How can this problem be solved for very low-function devices (such as network-attached RAM)?

By "macroscopic sharing" I mean "device reservations" or the like – allocating the entire (or a large chunk) of the device to a dedicated use, as opposed to attempting to coordinate access on an operation-by-operation basis (including caching). This is one area where the All-IP project has already done a little real work [64].

Here's an idea I've floated in a meeting or two:

⇒ Incorporate distributed resource management into a virtual machine monitor, such as Xen.

I'm sure VMware is headed in that direction; I don't know enough about VMotion and the other aspects of their remote device sharing to say how far they've gotten, but there should be room for exploration.

6.6 Security in Shared Devices

I don't believe that this area is at all yet mined out. Who has the right to grant access to a device or a piece of data? That is a generic systems question, but separation of security *policy* and *enforcement* can allow different policies in different situations. Should access rights (for either resource scheduling or security purposes) be persistent, or dynamic?

⇒ Implement derived virtual devices for DRAM.

Recent work has examined a form of "hardware firewall" for memory accesses in network-on-chip devices (really a simple MMU on a per-core basis) [30].

6.7 Speculation in Data Access

⇒ Presending of information based on various pieces of information. A good rule-based system is required.

It might be worthwhile to look at e.g. `lcron`, which allowed some location-specific computation [38].

And, if you're interested in prefetching, you gotta know about local prefetching and some of the intelligent methods that have been tried for that. I recommend Kroeger for both local FS and remote web prefetching [50, 49], and Korner [47] and Patterson [66] for additional ideas on more intelligent I/O systems, as well as rdv's SLEDs [85] and Arpaci-Dusseau's "gray box" techniques [8].

6.8 Speculation in Computation

This is the home run problem. Radical, innovative solutions here just *might* bring big improvements in system performance, and will definitely result in interesting papers.

Is it possible to predict a value or action accurately often enough to accelerate a computation?

Thread-level speculation is an active area of research [18]. There are many papers on the topic...

7. How to Have Fun Doing It

- Collaborate.
- Compete (in a friendly way).
- Set small, short-term goals with visible milestones.
- Work smart, not hard.
- Okay, work hard, too.

In my experience, success breeds success. When you are good, people want to work with you; when you accomplish one thing, it makes you feel good and makes you want to achieve even more.

8. References

[1] ACM. *Proc. 15th ACM Symposium on Operating Systems Principles*, Dec. 1995.

[2] ACM. *Proc. 16th ACM Symposium on Operating Systems Principles*, Oct. 1997.

[3] J. F. Adam, H. H. Houh, M. Ismert, and D. L. Tennenhouse. Media-intensive data communications in a "desk-area" network. *IEEE Communications*, pages 60–67, Aug. 1994.

[4] A. Al Zain, P. Trinder, G. Michaelson, and H. Loidl. Evaluating a High-Level Parallel Language (GpH) for Computational Grids. *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, pages 219–233, 2008.

[5] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The Globus Striped GridFTP Framework and Server. *Proceedings of Super Computing*, 2005, 2005.

[6] D. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *5th IEEE/ACM International Workshop on Grid Computing*, pages 365–372, 2004. Available translated into Japanese at http://boinc.oocp.org/grid_paper_04.html.

[7] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang. Serverless network file systems. In *Proc. 15th ACM Symposium on Operating Systems Principles* [1], pages 109–126.

[8] A. Arpaci-Dusseau and R. Arpaci-Dusseau. Information and control in gray-box systems. *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 43–56, 2001.

[9] R. Avnur and J. Hellerstein. Eddies: continuously adaptive query processing. *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 261–272, 2000.

[10] P. Barham, M. Hayter, D. McAuley, and I. Pratt. Devices on the desk area network. *J. Selected Areas in Communications*, 13(4):722–732, May 1995.

[11] A. Birrell and B. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems (TOCS)*, 2(1):39–59, 1984.

[12] T. Bjerregaard and S. Mahadevan. A survey of research and practices of Network-on-chip. *ACM Computing Surveys (CSUR)*, 38(1), 2006.

[13] P. Cao, S. B. Lim, S. Venkataraman, and J. Wilkes. The TickerTAIP parallel RAID architecture. In *Proc. 20th Annual International Symposium on Computer Architecture*, pages 52–63, May 1993.

[14] N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, 1989.

[15] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *Proc. Third Symposium on Operating Systems Design and Implementation*, Feb. 1999.

[16] D. Cheriton. The V distributed system. *Communications of the ACM*, 31(3), 1988.

[17] D. R. Cheriton. Exploiting recursion to simplify rpc communication architectures. In *Proc. ACM SigComm (Stanford, CA August 1988)*, pages 76–87, Aug. 1988.

[18] C. B. Colohan, A. Ailamaki, J. G. Steffan, and T. C. Mowry. Incrementally parallelizing database transactions with thread-level speculation. *ACM Trans. Comput. Syst.*, 26(1):1–50, 2008.

[19] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Addison-Wesley, 2nd edition, 1994.

[20] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[21] G. Delp, A. Sethi, and D. Farber. An analysis of MemNet—an experiment in high-speed shared-memory local networking. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 165–174, New York, NY, USA, 1988. ACM.

[22] J. Dennis. Data Flow Supercomputers. *Computer*, 13(11):48–56, 1980.

[23] F. Douglis and J. Ousterhout. Transparent process migration: design alternatives and the sprite implementation. *SoftwarePractice & Experience*, 21(8):757–785, 1991.

[24] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, and I. Pratt. Xen and the Art of Virtualization. *Proceedings of the ACM Symposium on Operating Systems Principles, October*, 2003.

[25] A. L. Drapeau, K. W. Shirrif, J. H. Hartman, E. L. Miller, S. Seshan, R. H. Katz, K. Lutz, D. A. Patterson, E. K. Lee, P. H. Chen, and G. A. Gibson. RAID-II: a high-bandwidth network file server. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 234–244, 1994.

[26] M. R. Eskicioglu. A comprehensive bibliography of distributed shared memory. *ACM Operating Systems Review*, pages 71–96, Jan. 1996.

[27] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, U. C. Irvine, 2000.

[28] R. Figueiredo, P. A. Dinda, and J. Fortes. Resource virtualization renaissance. *IEEE Computer*, pages 28–31, May 2005.

[29] G. G. Finn and P. Mockapetris. Netstation architecture: Multi-gigabit workstation network fabric. In *Proc. NetWorld+InterOp Engineer Conference*, 1994.

[30] L. Fiorin, G. Palermo, S. Lukovic, V. Catalano, and C. Silvano. Secure Memory Accesses on

Networks-on-Chip. *Computers, IEEE Transactions on*, 57(9):1216–1229, 2008.

[31] S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In *Proc. Symp. on Operating System Principles*, pages 29–43. ACM Press New York, NY, USA, 2003.

[32] G. A. Gibson et al. File server scaling with network-attached secure disks. In *Proc. ACM International Conference on Measurement and Modeling of Computer Systems*. ACM, June 1997.

[33] G. A. Gibson and R. Van Meter. Network attached storage architecture. *Communications of the ACM*, 43(11):37–45, Nov. 2000.

[34] C. Gray and D. Cheriton. Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. *SIGOPS Oper. Syst. Rev.*, 23(5):202–210, 1989.

[35] E. Hagersten, A. Landin, and S. Haridi. DDM-a cache-only memory architecture. *Computer*, 25(9):44–54, 1992.

[36] M. Harchol-Balter and A. B. Downey. Exploiting process lifetime distribution for dynamic load balancing. In *Proc. ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 13–24. ACM, May 1996.

[37] J. H. Hartman and J. K. Ousterhout. The zebra striped network file system. *ACM Trans. Comput. Syst.*, 13(3):274–310, Aug. 1995.

[38] J. Heidemann and D. Shah. Location-aware scheduling with minimal infrastructure. In *Proc. 2000 USENIX Annual Technical Conference*, pages 131–138.

[39] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufman, 4th edition, 2006.

[40] M. Huhns and M. Singh. Service-Oriented Computing: Key Concepts and Principles. *IEEE INTERNET COMPUTING*, pages 75–81, 2005.

[41] R. Jain, J. Werth, and J. C. Browne, editors. *Input/Output in Parallel and Distributed Computer Systems*. Kluwer Academic Publishers, 1996.

[42] D. Jefferson, B. Beckman, F. Wieland, L. Blume, M. Di Loreto, P. Hontalas, P. LaRouche, K. Sturdevant, J. Tupman, V. Warren, J. Wedel, H. Younger, and S. Bellenot. Distributed simulation and the Time Warp operating system. In *Proc. Eleventh ACM Symposium on Operating Systems Principles*, pages 77–93, Nov. 1987.

[43] R. H. Katz. High-performance network and channel based storage. *Proc. IEEE*, 90(8):1238–1261, Aug. 1992.

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

[45] M. Kim, L. Cox, and B. Noble. Safety, visibility, and performance in a wide-area file system. In *Proceedings of the Conference on File and Storage Technologies* [82].

[46] B. Kobler, editor. *Fifth NASA Goddard Conference on Mass Storage Systems and Technologies*, Sept. 1996.

[47] K. M. Korner. *An intelligent remote file server*. PhD thesis, 1986.

[48] C. E. Kozyrakis, S. Perissakis, D. A. Patterson, T. E. Anderson, K. Asanovic, N. Cardwell, R. Fromm, J. Golbus, B. Gribstad, K. Keeton, R. Thomas, N. Treuhaft, and K. A. Yelick. Scalable processors in the billion-transistor era: Iram. *IEEE Computer*, 30(9):75–78, 1997.

[49] T. M. Kroeger and D. D. E. Long. Predicting file-system actions from prior events. In *Proceedings of the USENIX 1996 Annual Technical Conference*, pages 319–328. USENIX, January 1996.

[50] T. M. Kroeger, D. D. E. Long, and J. C. Mogul. Exploring the bounds of Web latency reduction from caching and prefetching. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (ITS-97)*, pages 13–22, Berkeley, Dec. 8–11 1997. USENIX Association.

[51] N. P. Kronenberg, H. M. Levy, and W. D. Strecker. Vaxclusters: A closely-coupled distributed system. *ACM Trans. Comput. Syst.*, 4(2):130–146, May 1986.

[52] J. Kubiatowicz et al. OceanStore: An architecture for global-scale persistent storage. In *Proc. ACM Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, ACM, Nov. 2000.

[53] B. W. Lampson. Hints for computer system design. In *Proc. 9th ACM Symposium on Operating Systems Principles*, pages 33–48, 1983.

[54] J. Larus and C. Kozyrakis. Transactional memory. *Commun. ACM*, 51(7):80–89, 2008.

[55] E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *Proc. ACM Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 84–92. ACM, Oct. 1996.

[56] E. Levy and A. Silberschatz. Distributed file systems: Concepts and examples. *ACM Comput. Surv.*, 22(4):322–374, Dec. 1990.

[57] C. J. Lindblad, D. J. Wetherall, W. F. Stasior, J. F. Adam, H. H. Houh, M. Ismert, D. R. Bacher, B. M. Philips, and D. L. Tennenhouse. Viewstation applications: Implications for network traffic. *J. Selected Areas in Communications*, 13:768–778, May 1995.

[58] M. Litzkow, M. Livny, and M. Mutka. Condor-a hunter of idle workstations. *Distributed Computing Systems, 1988., 8th International Conference on*,

pages 104–111, 1988.

[59] H. Loidl, F. Rubio, N. Scaife, K. Hammond, S. Horiguchi, U. Klusik, R. Loogen, G. Michaelson, R. Peña, S. Priebe, et al. Comparing Parallel Functional Languages: Programming and Performance. *Higher-Order and Symbolic Computation*, 16(3):203–251, 2003.

[60] D. D. E. Long, B. R. Montague, and L.-F. Cabrera. Swift/RAID: A distributed RAID system. *Computing Systems*, 7(3):333–359, 1994.

[61] G. Memik, M. Kandemir, and A. Choudhary. Exploiting inter-file access patterns using multi-collective i/o. In *Proceedings of the Conference on File and Storage Technologies* [82].

[62] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile application-aware adaptation for mobility. In *Proc. 16th ACM Symposium on Operating Systems Principles* [2], pages 276–287.

[63] M. Nuttall. A Brief Summary of Systems Providing Process or Object Migration Facilities. *Operating Systems Review*, 28:64–80, 1994.

[64] K. Okada, K. Muda, Y. Nishida, H. Yoshifuji, R. Wakikawa, and J. Murai. Protocol Design for All-IP Computer Architecture. In *Information Networking, 2008. ICOIN 2008. International Conference on*, pages 1–5, 2008.

[65] J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson, and B. Welch. The Sprite Network Operating System. *Computer*, 21(2):23–36, 1988.

[66] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In *Proc. 15th ACM Symposium on Operating Systems Principles* [1], pages 79–95.

[67] R. Pike, K. Thompson, and H. Trickey. Plan 9 from Bell Labs. In *Proc. Summer 1990 UKUUG Conf.*, pages 1–9, July 1990.

[68] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, 1974.

[69] E. Riedel, C. Faloutsos, G. Gibson, and D. Nagle. Active disks for large-scale data processing. *Computer*, 34(6):68–74, 2001.

[70] E. Riedel and G. Gibson. Understanding customer dissatisfaction with underutilized distributed file servers. In Kobler [46], pages 371–388. also known as CMU-CS-96-158.

[71] R. H. Saavedra, R. S. Gaines, and M. J. Carlton. Micro benchmark analysis of the KSR1. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, pages 202–213, 1993.

[72] M. W. Sachs, A. Leff, and D. Sevigny. LAN and I/O convergence: A survey of the issues. *IEEE Computer*, pages 24–32, Dec. 1994.

[73] P. Sarkar, S. Uttamchandani, and K. Voruganti. Storage Over IP: When Does Hardware Support Help? *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 231–244, 2003.

[74] M. Satyanarayanan. Scalable, secure, and high available distributed file access. *IEEE Computer*, pages 9–21, May 1990.

[75] J. Smith. A survey of process migration mechanisms. *Operating systems review*, 22(3):28–40, 1988. note: more than one version of this paper exists.

[76] S. R. Soltis, T. M. Ruwart, and M. T. O'Keefe. The global file system. In Kobler [46], pages 319–342.

[77] M.-C. Tam, J. M. Smith, and D. J. Farber. A taxonomy-based comparison of several distributed shared memory systems. *SIGOPS Oper. Syst. Rev.*, 24(3):40–67, 1990.

[78] A. S. Tanenbaum, R. van Renesse, H. van Stavaren, G. J. Sharp, S. J. Mullender, J. Jansen, and G. van Rossum. Experiences with the Amoeba distributed operating system. *Commun. ACM*, 33(12):46–63, Dec. 1990.

[79] C. A. Thekkath, T. Mann, and E. K. Lee. Frangipani: A scalable distributed file system. In *Proc. 16th ACM Symposium on Operating Systems Principles* [2], pages 224–237.

[80] J. Touch and D. Farber. Memory-side driven anticipatory instruction transfer interface with processor-side instruction selection, Oct. 4 1994. US Patent 5,353,419.

[81] J. D. Touch. *MIRAGE: A Model for Latency in Communication*. PhD thesis, University of Pennsylvania, 1992.

[82] USENIX. *Proceedings of the Conference on File and Storage Technologies*. USENIX, Jan. 2002.

[83] M. Uysal, A. Acharya, and J. Saltz. Evaluation of active disks for decision support databases. *Proc. Sixth Intl Symp. High-Performance Computer Architecture*, pages 337–348, 2000.

[84] R. Van Meter. A brief survey of current work on network attached peripherals (extended abstract). *ACM Operating Systems Review*, pages 63–70, Jan. 1996.

[85] R. Van Meter and M. Gao. Latency management in storage systems. In *Proc. Fourth USENIX Symp. on Operating Systems Design and Implementation*. USENIX, Oct. 2000.

[86] R. Van Meter, S. Hotz, and G. Finn. Derived virtual devices: A secure distributed file system mechanism. In Kobler [46].

[87] B. Walker, G. Popek, R. English, C. Kline, and G. Thiel. The LOCUS distributed operating system. In *Proceedings of the ninth ACM symposium on operating systems principles*, pages 49–70. ACM New York, NY, USA, 1983.