New wine in old skins: the case for distributed operating systems in the data center

Malte Schwarzkopf [†] Matthew P. Grosvenor[†] Steven Hand[†‡]
[†]*University of Cambridge Computer Laboratory* [‡]*Microsoft Research Silicon Valley*

The argument, 'that you remember exactly the same proposal being rejected in 1867,' is a very strong one in itself, but its defect is that it appeals only to those who also remember the year 1867 with affectionate interest, and, moreover, are unaware that any change has occurred since then.
— F.M. Cornford,
"Microcosmographia Academica", 1908.

Abstract

Since their heyday the 1980s, distributed operating systems—spanning multiple autonomous machines, yet appearing to the user as a single machine—have seen only moderate academic interest. This is a little surprising, since modern data centers might present an appealing environment for their deployment. In this position paper, we discuss what has changed since the community lost interest in them, and why, nonetheless, distributed OSes have yet to be considered for data centers. Finally, we argue that the distributed OS concept is worth a revisit, and outline the benefits to be had from it in the context of the modern data center.

1 Introduction

Distributed programming abstractions are as old as the ability to connect computers together [42, 43]. Nonetheless, historically the size and cost of owning a single computer for a long time restricted the dominant model to dumb terminals connecting to a single central, time-shared machine. In the 1980s, however, cheap personal computers with fast and standardized networking technology for LANs became widely available, and triggered a wave of research into distributed operating systems. While the differentiation between distributed systems and distributed *operating* systems was fuzzy at the time, later literature typically defines a distributed OS as *(i)* spanning multiple independent, autonomous and communicating CPUs, and *(ii)* appearing towards the user as a single machine [40]. Many such systems were developed [12, 22, 23, 25, 30, 32, 34, 38], and by the late 1980s, it was widely expected that distributed operating systems would soon be ubiquitous [40].

Nothing could have been further from the truth. The 1990s became the decade of desktop computing, dominated by traditional single machine OSes with their roots in time-shared mainframe systems (UNIX, BSDs, Linux) and systems based on newly developed home microcomputer OSes (such as DOS or MacOS). Distributed operation happened at the application level, based on a client-server model, and with the machine boundary being impenetrable to the OS and its abstractions. Upon the client-server model, the rise of internet-based services proliferated a hosted application model that has culminated in the "cloud" paradigm of recent years.

To support the server side of such applications, large-scale data centers are both prevalent and indispensable nowadays. Due to economies of scale, these are typically built as clusters of many computers from commodity parts [5, 18]. Nevertheless, the demands of the large-scale applications running within them necessitate the use of many machines for a single application, and the desire for simplicity in programming and managment models yields the common abstraction of the data center as a single "warehouse-scale computer" (WSC) [18]. This setup, curiously, constitutes exactly the environment that distributed OS designers had in mind: *a set of independent computers that appears as a single system to its users*.

One might be tempted to expect this development to have rekindled interest in distributed OS research. Yet this has not happened. In this position paper, we first dis-

cuss why we believe the distributed OS concept did not originally succeed (§2) and what has changed since (§3). Despite those changes, distributed OS research remains a niche interest, and in §4 we speculate why. Arguing that, nonetheless, the time is right for a comeback, we explain what benefits are to be had from thinking about distributed OSes in the context of modern WSC data centers (§5). We finish by highlighting useful concepts for a distributed WSC OS and debate the resulting challenges (§6), before concluding (§7).

2 An idea ahead of the times

In the early 1980s, much enthusiasm erupted for designing fundamentally *distributed* computing systems [24]. A common concept in these research systems was the idea of sharing resources on a LAN of workstations, giving users the additional flexibility to use remote compute power when necessary, and transparently offering networked services for data storage, printing and communication. This flexibility, however, came at a cost.

The cost of transparency. The philosophical difference between the same services implemented atop traditional time-shared OSes and a distributed OS lies in the fact that the latter performs distribution *transparently* to the user [40], who is completely ignorant of *where* processes and data are. This very advantage turned out to be a key nail in the coffin of distributed OSes' commoditization, however: as desktop applications evolved rapidly, transparent distributed OS abstractions such as distributed shared virtual memory (DSVM) [17] and process migration [32] turned out to be a hindrance to deterministic performance. This was problematic for interactive desktop applications, as their resource and responsiveness needs exceeded the guarantees available from the networking technology and software fault-tolerance of the time. For an application running atop a distributed OS with DSVM, any of its pages could be resident on any machine, and its execution threads might be migrated as the OS sees fit. At a multi-millisecond cost [17, 23] for each remote page acquire and write operation (which may occur non-deterministically at any time!), it is not surprising that programmers and users preferred the safe performance determinism of traditional OSes.

Compute/communication speed dichotomy. One reason why the cost of remote operations was felt so dearly is that microprocessor clock speeds and bottom line performance increased rapidly in the 1980s and 1990s—far more so than communication speeds. While LAN bandwidths of 10 MBit/s were available by the mid-1980s, CPU clock rates of 10-20 MHz on RISC architectures meant that compute speed far exceeded communication speed. The 1990s only accelerated this trend: as clock rates raced upwards and eventually

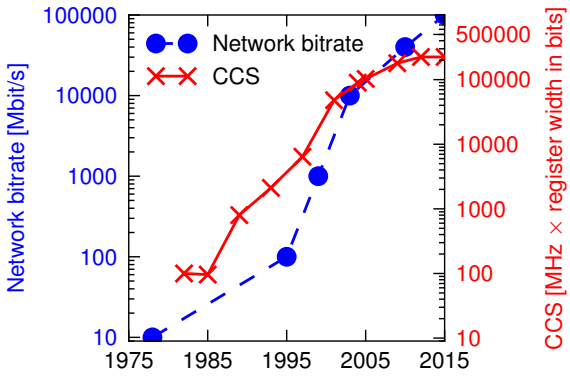

Figure 1: LAN communication bandwidth and Comparative Compute Speed (CCS; clock speed in MHz × register width in bits*) over time (N.B.: log-scale y-axes).*

peaked around 3.5 GHz for thermal reasons in the early 2000s, LAN bandwidth scaling largely lagged behind serial per-CPU compute performance. Figure 1 illustrates this trend: it shows the Comparative Compute Speed (CCS) of a range of Intel CPUs and the maximum standard Ethernet bitrates over the period from 1982 to 2015.[1] At the same time, communication latency only improved by very little [33, 39], widening the gap between computation and communication speed. While bandwidth has now started to outscale compute speed (cf. Figure 1), unmasked latency still has a high cost. However, as we explain in §3, many modern WSC applications fundamentally must be distributed (unlike the earlier desktop applications!), and thus already have to deal with the latency issue.

Conflation of micro-kernels and distributed systems. Many distributed OS research projects chose a micro-kernel as a central design component, aiming to run as little code as possible in privileged mode. This made sense: as micro-kernels provide communication primitives for IPC, it appeared logical to simply extend such mechanisms to support network communication (e.g. via RPCs). Moreover, micro-kernels enforce a disaggregation of the OS into independent components—a view highly compatible with the distributed OS vision, enabling components to run in different places. The choice of micro-kernels as a basis, however, put distributed operating systems at a further disadvantage in the competitive arena of 1990s desktop computing, as they also inherited many perceived drawbacks of the micro-kernel approach [16].

[1] CCS is *clock speed in MHz* × *register width in bits*, for fairer comparison to Ethernet bitrates.

3 How the world has changed

The historical evidence clearly shows the failure of the distributed OS concept. Instead, orchestration of distributed operation is now firmly the business of applications. So, can we declare the case closed? Not quite, as some key conditions have recently changed, or are in the process of changing.

I/O is faster than computation. While CPU clock rates have plateaued in recent years, network communication bandwidth keeps increasing:[2] A typical CPU core nowadays runs at 2-3 GHz and increases in serial CPU speed have stagnated, while data center LANs are adopting 10 GBit/s Ethernet, with 40 GBit/s and 100 GBit/s Ethernet on the horizon. Indeed, long-standing OS abstractions designed on the premise that network I/O is far slower than local operations—such as the BSD sockets interface—do not scale to ever-faster networks [15]. Of course, the aggregate compute capacity is still increasing as the number of cores per processor keeps growing. The need to exploit this parallelism, however, brings even single-machine abstractions closer to distributed systems [6]. Due to this reversal of relative speed, the performance advantage of spatial locality is lost at least for some applications. Indeed, it has been observed that accessing remote memory and disks has no substantial overhead any more [3, 31]. As a second-order consequence, moving computation and state (e.g. towards a faster processor or a GPGPU/FPGA accelerator, or away from contention) is increasingly viable, as migration overheads diminish.

Data center applications necessitate distribution. Large-scale WSCs in data centers were not conceivable in the 1980s, although notions of much smaller-scale shared "processor banks" [27] and "processor pools" [25] existed. Modern WSC applications are sufficiently demanding that a single machine alone often cannot satisfy their needs. They *require* distribution for scale and fault-tolerance, rather than merely benefiting from the flexibility granted. Thus, existing data centre application stacks implement many of the features originally supported in distributed operating systems as user-space "middleware" libraries (e.g. name services, distributed locking, caching and scheduling). The role of the traditional OS, with its myopic view of the local machine, on the other hand, is marginalized.

Micro-architectures meet distributed systems. Modern multi-socket many-core machines have complex internal interconnects as well as deep cache hierarchies. Hence, even the within-chassis environment is increasingly reminiscent of a networked distributed system, and operating systems must adapt to this new reality [8]. Indeed, some predict that cache-coherent shared memory is doomed as the number of CPU cores increases, and will give way to message-based network-on-chip communication [4]. As shared memory and message passing are dual in expressivity [21], a distributed OS can ideally map a single system view on top of such a messaging layer with little extra cost over that which is already incurred due increasingly distributed hardware.

Transparency is in demand. Distributed programming for the analysis of large data sets and the deployment of complex distributed services has become a mainstream activity in WSCs. For this purpose, the most popular systems are those frameworks that raise the level of abstraction and liberate the programmer from knowing the details of distributed coordination, synchronization and fault-tolerance implementations. The emergence of MapReduce [14] and many similar computing frameworks [19, 26, 44], alongside infrastructure systems for distributed storage [11, 29], are testimony to this trend. Since these frameworks already provide transparent distribution and tolerate the overheads of coarse-grained parallelism by design, they are a better fit for a distributed OS than the earlier desktop applications.

In summary, the environment has changed dramatically since the 1990s; a revisit of distributed OS ideas seems overdue against the backdrop of WSCs. Yet, it has not happened—in the following, we discuss why.

4 Why distributed OSes have not been revisited yet

WSCs are a particular, specialized distributed environment of unprecedented scale: a data center with 20,000 servers of 48 cores each (i.e., a million cores) is now a plausible setup [36]. But size is not the only distinguishing factor: WSCs, unlike traditional data centers, are operated by (and for) a single organization and "much of the application, middleware, and system software is built in-house" [5, 18]. At the OS level, WSCs today run many instances of a commodity microcomputer OS linked together by user-space "middleware" [41]. Why, given the apparent similarity to the target environment of distributed OSes, has nobody considered designing a distributed OS for a WSC? We believe there are four key reasons.

1. **Legacy code support.** The success of open source software has produced a giant corpus of legacy code written on top of the UNIX/POSIX paradigms—from device drivers to libraries and applications. Ignoring this legacy, or breaking compatibility with it, seems like an unwise decision. Much distributed systems software has been written atop the sockets abstraction and thus internet protocols have become a lowest-

[2] Latency, however, does not improve similarly, as we discuss in §6.

common denominator communication mechanism in WSCs, despite not always being optimal choices. This is especially true in the case of TCP, which is designed for much more pessimal network conditions than those typically found in WSCs [2].

2. **Migration inertia.** Current data center execution environments are identical to those we use on personal machines. This yields immediate developer familiarity as an advantage, but also creates a strong inertia to stick to known principles and software packages. While a new high-level framework API for distributed data processing may be easy for a programmer to pick up, the potential learning curve for a different OS is much steeper.

3. **Rapid evolution.** WSCs evolved and became prevalent in response to the business needs of rapidly growing internet companies. The main challenge for a long time was to scale up as fast as possible, causing rapid evolution of software platforms and frameworks alike. This pace of innovation is well beyond even the most optimistic OS development timescales: traditional user-space software development and debugging is easier, cheaper and more readily utilizes existing tools than in-kernel development or working at the OS level.

4. **Focus on multi-core and virtualization.** The OS research community, on the other hand, while unencumbered by immediate business imperatives, has focused elsewhere. In particular, much research attention has been dedicated to virtualization and support for heterogeneous multi-core machines [6, 28]. Virtualization injects a layer of abstraction below the traditional machine boundary, but expressly does not offer transparency or a single-system view across multiple VMs to applications. Heterogeneous multi-core research does try to offer such a view, but is focused on bridging architectural and instruction set gaps, rather than extending the scope of the OS across machines. As a consequence, researchers have not spent as much time considering how the OS in a WSC can be specialized or redesigned, or how OS primitives can be extended across machine boundaries.

5 Why the time is right

We do not believe that any of the reasons presented in the previous section are fundamental. Multi-core hardware featuring vastly increased parallelism and increasingly asymmetric performance characteristics is already forcing code to be re-engineered [10]. Furthermore it is often the case that WSC operators extensively modify the OS: in 2009, Google were already maintaining 1,280 proprietary patches adding ~300,000 lines to the Linux kernel [41] for their WSC operations. In this light, we believe that the time has come to reconsider distributed OS research, and in the following outline the benefits of doing so.

Automated decisions. The complexity of WSC-scale operations and applications is such that key operating decisions are best made by integrated software stacks. At the same time, simplicity in programming and configuration abstractions is of key importance. Distributed OSes have the potential to provide both: the OS can decide where, when and what to execute, while permitting the user to provide, and reason about, high-level intentions instead of minute details of mechanism. Why must this take place in the OS, rather than continuing in user-space "middleware"? While we are aware that there is a delicate balance between feature expansion and attack surface expansion [20], we believe pushing distributed systems functionality into shared OS components can be beneficial. As with all operating systems, common operations be implemented once and shared between all applications in the WSC, and applications can assume them to be present and trustworthy. Furthermore, additional efficiency and security enhancements are only possible when the per-node kernel is fully aware of its role as a component in a massively distributed system.

Information flow control and provenance. Controlling and managing the provenance of user data is a key challenge in WSC environments. Forcing applications to access data and communicate using the abstractions of a distributed OS provides a convenient mechanism for tracking information flow. Thus, a distributed WSC OS can provide assurances about data management policy conformance (and detect violation attempts). This is notoriously difficult in the user-space of a traditional OS, where it is possible to bypass such mechanisms trivially. Given the opportunity to re-architect the OS to match its operating environment, this functionality need not come with severe performance penalties.

Distributed tracing and debugging. The same interception mechanisms described above can also be leveraged for purposes of debugging and tracing WSC application software—a well known and challenging problem with traditional WSC setups [9] due to the additional complexity introduced by the mismatch between OS and distributed systems abstractions.

Opportunities for new abstractions. Traditional OS abstractions for I/O, resource allocation and isolation were not designed for the highly loaded, densly shared environment of a WSC. Reconsidering OS design for the WSC presents opportunities for better tuned abstractions. For example, *recursive abstractions* may permit applications to be ignorant of whether they are operating on a slice of a machine, the cluster, or indeed a sub-slice of an

existing slice. Distributed OSes can excel at providing such a view, since they permit interposition of operations for distribution. Furthermore, increased networking and disk I/O performance have highlighted limitations in traditional interfaces (such as sockets). Workarounds for these exist [37], but often have drawbacks—such as lack of protection due to bypassing the kernel or limitations on concurrent access. The highly transparent design of a distributed OS allows a different approach: the OS can take care of the precise I/O mechanism, while the application sees only a high level API.

6 Concepts and challenges

In the following, we briefly discuss key concepts and challenges for distributed WSC OS, and point to existing work that might help addressing them.

Unified naming. Workflows in WSCs require data to be uniquely identified, so that applications can transparently check their existence and retrieve them from anywhere. Traditional page-level distributed shared memory may be too fine-grained for this purpose. Approaches akin to DOMAIN's "object storage system" [23], which provides unique IDs to all objects from memory regions to hardware devices, might be worth investigating.

Data-flow abstractions. Distributed data-flow is a popular choice for WSC application programming as it is amenable to exposing a transparent high-level API [19, 26]. While traditional distributed OSes tend to stick closely to the familiar model of files and processes, a future WSC OS may benefit from being built around data-flow abstractions. This is particularly pertinent given that data-flow maps well onto capability-based security mechanisms, and has attractive properties (e.g. explicit dependencies) for information flow tracking.

Network latency. While network bandwidth has outscaled serial CPU performance in recent years, network latency has not reduced by nearly as much [33, 39]. Indeed, there is some evidence that it is only getting worse [5, 13]. This has a direct impact on the performance of a distributed WSC OS. As we have argued, WSC applications may tolerate this, since they are already subject to network latencies by virtue of being distributed, but improvements are still desirable. Good engineering can help, but the answer may lie in old concepts such as ordered broadcast [40]: increased bandwidth makes time-division multiplexing of a fraction of the link capacity a viable optimization for low latency.

Backwards compatibility. It would seem that despite all the benefits, a distributed WSC OS would still suffer the Achilles heel of breaking legacy code compatibility by changing OS APIs and semantics. Traditional distributed OSes handle this through emulation libraries [1, 38] or syscall interposition [25, 40]. On a WSC, we can run both a traditional OS and a new distributed OS side-by-side (at least for incremental migration). Recent work on OS extensibility and library OSes shows how legacy OSes can be run on top of adaptation layers at low overhead [7, 35], and applications already ported to the distributed WSC OS can run on machines with the legacy OS via a shim library implementing the distributed OS API in user-space.

Distributed consistency. Most traditional distributed OSes use single ownership models for distributed consistency. Since the 1980s, however, weaker consistency models for higher availability have received much attention. As these are particularly popular in data center applications, it seems apt to consider whether a distributed WSC OS could depart from the traditional, strongly-consistent state in state in distributed OSes. Having OS-level support for for relaxed consistency operations—both for application I/O and for OS state—may help masking inevitable communication latency and increase overall scalability.

Security considerations. The 1980s distributed operating systems had—by today's standards—fairly naïve security and authentication models, and much work in distributed systems security has followed since. Due to the sensitive nature of data processed on WSCs, it is paramount that a distributed WSC OS is able to prevent accidental data disclosure, security compromises and denial-of-service attacks to the furthest possible extent. Hence, a revisit of distributed OSes for the data center will likely have to dedicate significant attention to ways of ensuring integrity and security are maintained. Capability-based models might be an especially good candidate to investigate here, as they fit naturally with the data-flow nature of data center software.

7 Discussion and conclusion

A quarter century down the line, distributed operating systems are due for a revisit. This time, providing a special-purpose OS well suited to WSCs should be the goal—not flexible desktop computing. Indeed, we believe that any special-purpose data center OS will necessarily have to acknowledge and build upon the breadth work done on distributed OSes in the past. As such, we have put our money where our mouth is, and are developing on DIOS, a new distributed operating system design for WSCs. A prototype implementation is in the works.

8 Acknowledgements

Special thanks go to the members of the Cambridge operating systems reading group and our colleagues in the wider Systems Research Group, whose feedback much improved the clarity of our vision: Frank Bellosa, David Chisnall, Natacha Crooks, Ionel Gog, Anil Mad-

havapeddy, Peter Neumann and Robert N. M. Watson.

This work was partially supported by the EPSRC INTERNET Project EP/H040536/1 and the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-11-C-0249. The views, opinions, and/or findings contained in this article/presentation are those of the author/ presenter and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense.

References

[1] ACCETTA, M., BARON, R., BOLOSKY, W., GOLUB, D., RASHID, R., TEVANIAN, A., AND YOUNG, M. Mach: A new kernel foundation for UNIX development. In *Proceedings of Summer USENIX Conference* (1986).

[2] ALIZADEH, M., GREENBERG, A., MALTZ, D. A., PADHYE, J., PATEL, P., PRABHAKAR, B., SENGUPTA, S., AND SRIDHARAN, M. Data Center TCP (DCTCP). *Computer Communication Review 40*, 4 (2010), 63–74.

[3] ANANTHANARAYANAN, G., GHODSI, A., SHENKER, S., AND STOICA, I. Disk-Locality in Datacenter Computing Considered Irrelevant. In *Proceedings of HotOS* (2011).

[4] BARON, M. The Single-Chip Cloud Computer. *Microprocessor Report 24*, 4 (2010).

[5] BARROSO, L. A. Warehouse-scale computing: Entering the teenage decade. *SIGARCH Computer Architecture News 39*, 3 (June 2011).

[6] BAUMANN, A., , BARHAM, P., DAGAND, P.-E., HARRIS, T., ISAACS, R., PETER, S., ROSCOE, T., SCHÜPBACH, A., AND SINGHANIA, A. The multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of SOSP* (2009).

[7] BAUMANN, A., LEE, D., FONSECA, P., GLENDENNING, L., LORCH, J. R., BOND, B., OLINSKY, R., AND HUNT, G. C. Composing OS extensions safely and efficiently with Bascule. In *Proceedings of EuroSys* (2013).

[8] BAUMANN, A., PETER, S., SCHÜPBACH, A., SINGHANIA, A., ROSCOE, T., BARHAM, P., AND ISAACS, R. Your computer is already a distributed system. Why isn't your OS? In *Proceedings of HotOS* (2009).

[9] BLIGH, M., DESNOYERS, M., AND SCHULTZ, R. Linux Kernel Debugging on Google-sized Clusters. In *Proceedings of the Ottawa Linux Symposium* (2007).

[10] BOYD-WICKIZER, S., CHEN, H., CHEN, R., MAO, Y., KAASHOEK, M. F., MORRIS, R., PESTEREV, A., STEIN, L., WU, M., DAI, Y.-H., ET AL. Corey: An operating system for many cores. In *Proceedings of OSDI* (2008).

[11] CHANG, F., DEAN, J., GHEMAWAT, S., HSIEH, W. C., WALLACH, D. A., BURROWS, M., CHANDRA, T., FIKES, A., AND GRUBER, R. E. Bigtable: A Distributed Storage System for Structured Data. In *Proceedings of OSDI* (2006).

[12] CHERITON, D. The V distributed system. *Commun. ACM 31*, 3 (Mar. 1988), 314–333.

[13] DEAN, J., AND BARROSO, L. A. The tail at scale. *Communications of the ACM 56*, 2 (Feb. 2013), 74–80.

[14] DEAN, J., AND GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of OSDI* (2004).

[15] HAN, S., MARSHALL, S., CHUN, B.-G., AND RATNASAMY, S. MegaPipe: a new programming interface for scalable network I/O. In *Proceedings of OSDI* (2012), pp. 135–148.

[16] HAND, S., WARFIELD, A., FRASER, K., KOTSOVINOS, E., AND MAGENHEIMER, D. J. Are virtual machine monitors microkernels done right? In *Proceedings of HotOS* (2005).

[17] HEISER, G. AND ELPHINSTONE, K. AND VOCHTELOO, J. AND RUSSELL, S. AND LIEDTKE, J. The Mungi single-address-space operating system. *Software: Practice and Experience 28*, 9 (1998), 901–928.

[18] HOELZLE, U., AND BARROSO, L. A. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 1st ed. Morgan and Claypool Publishers, 2009.

[19] ISARD, M., BUDIU, M., YU, Y., BIRRELL, A., AND FETTERLY, D. Dryad: distributed data-parallel programs from sequential building blocks. *Operating Systems Review 41*, 3 (June 2007), 59.

[20] KURMUS, A., SORNIOTTI, A., AND KAPITZA, R. Attack surface reduction for commodity OS kernels: trimmed garden plants may attract less bugs. In *Proceedings of EuroSec* (2011), pp. 6:1–6:6.

[21] LAUER, H. C., AND NEEDHAM, R. M. On the duality of operating system structures. *Operating Systems Review 13*, 2 (Apr. 1979), 3–19.

[22] LAZOWSKA, E. D., LEVY, H. M., ALMES, G. T., FISCHER, M. J., FOWLER, R. J., AND VESTAL, S. C. The architecture of the Eden system. In *Proceedings of SOSP* (1981).

[23] LEACH, P., LEVINE, P., DOUROS, B., HAMILTON, J., NELSON, D., AND STUMPF, B. The architecture of an integrated local network. *Selected Areas in Communications 1*, 5 (Nov. 1983), 842 – 857.

[24] LISKOV, B. Primitives for distributed computing. In *Proceedings of SOSP* (1979).

[25] MULLENDER, S., VAN ROSSUM, G., TANANBAUM, A., VAN RENESSE, R., AND VAN STAVEREN, H. Amoeba: a distributed operating system for the 1990s. *Computer 23*, 5 (May 1990), 44 –53.

[26] MURRAY, D. G., SCHWARZKOPF, M., SMOWTON, C., SMITH, S., MADHAVAPEDDY, A., AND HAND, S. CIEL: a universal execution engine for distributed data-flow computing. In *Proceedings of NSDI* (2011).

[27] NEEDHAM, R. M., AND HERBERT, A. J. *The Cambridge distributed computing system*. Addison Wesley Publish. Co., 1983.

[28] NIGHTINGALE, E. B., HODSON, O., MCILROY, R., HAWBLITZEL, C., AND HUNT, G. Helios: heterogeneous multiprocessing with satellite kernels. In *Proceedings of SOSP* (2009).

[29] NIGHTINGALE, E. AND ELSON, J. AND HOFMANN, O. AND SUZUE, Y. AND FAN, J. AND HOWELL, J. Flat Datacenter Storage. In *Proceedings of OSDI* (2012).

[30] NORTHCUTT, J. D. *Mechanisms for reliable distributed real-time operating systems: The Alpha Kernel*. Academic Press Professional, Inc., 1987.

[31] OUSTERHOUT, J., AGRAWAL, P., ERICKSON, D., KOZYRAKIS, C., LEVERICH, J., MAZIÈRES, D., MITRA, S., NARAYANAN, A., PARULKAR, G., ROSENBLUM, M., ET AL. The case for RAMClouds: scalable high-performance storage entirely in DRAM. *Operating Systems Review 43*, 4 (Jan. 2010), 92–105.

[32] OUSTERHOUT, J., CHERENSON, A., DOUGLIS, F., NELSON, M., AND WELCH, B. The Sprite network operating system. *Computer 21*, 2 (Feb. 1988), 23 –36.

[33] PATTERSON, D. A. Latency lags bandwith. *Communications of the ACM 47*, 10 (Oct. 2004), 71–75.

[34] POPEK, G., WALKER, B., CHOW, J., EDWARDS, D., KLINE, C., RUDISIN, G., AND THIEL, G. LOCUS: a network transparent, high reliability distributed system. In *Proceedings of SOSP* (1981).

[35] PORTER, D.E. AND BOYD-WICKIZER, S. AND HOWELL, J. AND OLINSKY, R. AND HUNT, G.C. Rethinking the library OS from the top down. In *Proceedings of ASPLOS* (2011), pp. 291–304.

[36] REISS, C. AND TUMANOV, A. AND GANGER, G.R. AND KATZ, R.H. AND KOZUCH, M.A. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of SoCC* (2012).

[37] RIZZO, L. netmap: a novel framework for fast packet I/O. In *Proceedings of USENIX ATC* (2012).

[38] ROZIER, M., ABROSSIMOV, V., ARMAND, F., BOULE, I., GIEN, M., GUILLEMONT, M., HERRMANN, F., KAISER, C., LANGLOIS, S., LÉONARD, P., ET AL. Overview of the CHORUS Distributed Operating Systems. *Computing Systems 1* (1991), 39–69.

[39] RUMBLE, S. M., ONGARO, D., STUTSMAN, R., ROSENBLUM, M., AND OUSTERHOUT, J. K. Its time for low latency. In *Proceedings of HotOS* (2011).

[40] TANENBAUM, A. S. Distributed operating systems anno 1992. What have we learned so far? *Distributed Systems Engineering. 1*, 1 (1993), 3.

[41] WAYCHINSON, M., AND CORBET, J. KS2009: How Google uses Linux, 2009. `http://lwn.net/Articles/357658/`.

[42] WEIK, M. H. A Third Survey of Domestic Electronic Digital Computing Systems. Tech. rep., Ballistic Research Laboratories, Mar. 1961.

[43] WILKES, M. *Memoirs of a computer pioneer*. MIT Press, 1985.

[44] ZAHARIA, M., CHOWDHURY, M., DAS, T., DAVE, A., MA, J., MCCAULEY, M., FRANKLIN, M., SHENKER, S., AND STOICA, I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of NSDI* (2012), p. 2.