A Cloud-Scale Acceleration Architecture

Adrian M. Caulfield Eric S. Chung Andrew Putnam
Hari Angepat Jeremy Fowers Michael Haselman Stephen Heil Matt Humphrey
Puneet Kaur Joo-Young Kim Daniel Lo Todd Massengill Kalin Ovtcharov
Michael Papamichael Lisa Woods Sitaram Lanka Derek Chiou Doug Burger

Microsoft Corporation

***Abstract*—Hyperscale datacenter providers have struggled to balance the growing need for specialized hardware (efficiency) with the economic benefits of homogeneity (manageability). In this paper we propose a new cloud architecture that uses reconfigurable logic to accelerate both network plane functions and applications. This Configurable Cloud architecture places a layer of reconfigurable logic (FPGAs) between the network switches and the servers, enabling network flows to be programmably transformed at line rate, enabling acceleration of local applications running on the server, and enabling the FPGAs to communicate directly, at datacenter scale, to harvest remote FPGAs unused by their local servers. We deployed this design over a production server bed, and show how it can be used for both service acceleration (Web search ranking) and network acceleration (encryption of data in transit at high-speeds). This architecture is much more scalable than prior work which used secondary rack-scale networks for inter-FPGA communication. By coupling to the network plane, direct FPGA-to-FPGA messages can be achieved at comparable latency to previous work, without the secondary network. Additionally, the scale of direct inter-FPGA messaging is much larger. The average round-trip latencies observed in our measurements among 24, 1000, and 250,000 machines are under 3, 9, and 20 microseconds, respectively. The Configurable Cloud architecture has been deployed at hyperscale in Microsoft's production datacenters worldwide.**

I. Introduction

Modern hyperscale datacenters have made huge strides with improvements in networking, virtualization, energy efficiency, and infrastructure management, but still have the same basic structure as they have for years: individual servers with multicore CPUs, DRAM, and local storage, connected by the NIC through Ethernet switches to other servers. At hyperscale (hundreds of thousands to millions of servers), there are significant benefits to maximizing homogeneity; workloads can be migrated fungibly across the infrastructure, and management is simplified, reducing costs and configuration errors.

Both the slowdown in CPU scaling and the ending of Moore's Law have resulted in a growing need for hardware specialization to increase performance and efficiency. However, placing specialized accelerators in a subset of a hyperscale infrastructure's servers reduces the highly desirable homogeneity. The question is mostly one of economics: whether it is cost-effective to deploy an accelerator in every new server, whether it is better to specialize a subset of an infrastructure's new servers and maintain an ever-growing number of configurations, or whether it is most cost-effective to do neither. Any specialized accelerator must be compatible with the target workloads through its deployment lifetime (e.g. six years: two years to design and deploy the accelerator and four years of server deployment lifetime). This requirement is a challenge given both the diversity of cloud workloads and the rapid rate at which they change (weekly or monthly). It is thus highly desirable that accelerators incorporated into hyperscale servers be programmable, the two most common examples being FPGAs and GPUs.

978-1-5090-3508-3/16/$31.00 © 2016 IEEE

Both GPUs and FPGAs have been deployed in datacenter infrastructure at reasonable scale without direct connectivity between accelerators [1], [2], [3]. Our recent publication described a medium-scale FPGA deployment in a production datacenter to accelerate Bing web search ranking using multiple directly-connected accelerators [4]. That design consisted of a rack-scale fabric of 48 FPGAs connected by a secondary network. While effective at accelerating search ranking, our first architecture had several significant limitations:

- The secondary network (a 6x8 torus) required expensive and complex cabling, and required awareness of the physical location of machines.
- Failure handling of the torus required complex re-routing of traffic to neighboring nodes, causing both performance loss and isolation of nodes under certain failure patterns.
- The number of FPGAs that could communicate directly, without going through software, was limited to a single rack (i.e. 48 nodes).
- The fabric was a limited-scale "bolt on" accelerator, which could accelerate applications but offered little for enhancing the datacenter infrastructure, such as networking and storage flows.

In this paper, we describe a new cloud-scale, FPGA-based acceleration architecture, which we call the *Configurable Cloud*, which eliminates all of the limitations listed above with a single design. This architecture has been — and is being — deployed in the majority of new servers in Microsoft's production datacenters across more than 15 countries and 5 continents. A Configurable Cloud allows the datapath of cloud communication to be accelerated with programmable hardware. This datapath can include networking flows, storage flows, security operations, and distributed (multi-FPGA) applications.

The key difference over previous work is that the accelera-

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

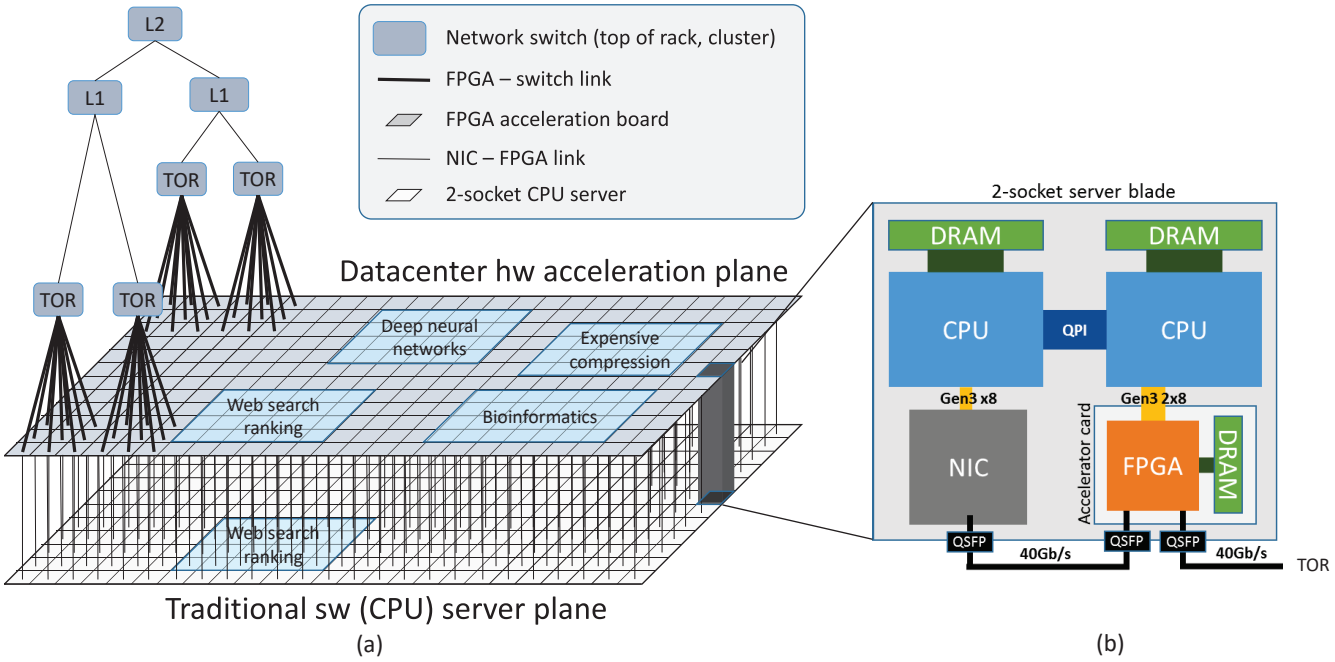

Fig. 1. (a) Decoupled Programmable Hardware Plane, (b) Server + FPGA schematic.

tion hardware is tightly coupled with the datacenter network—placing a layer of FPGAs between the servers' NICs and the Ethernet network switches. Figure 1b shows how the accelerator fits into a host server. All network traffic is routed through the FPGA, allowing it to accelerate high-bandwidth network flows. An independent PCIe connection to the host CPUs is also provided, allowing the FPGA to be used as a local compute accelerator. The standard network switch and topology removes the impact of failures on neighboring servers, removes the need for non-standard cabling, and eliminates the need to track the physical location of machines in each rack.

While placing FPGAs as a network-side "bump-in-the-wire" solves many of the shortcomings of the torus topology, much more is possible. By enabling the FPGAs to generate and consume their own networking packets independent of the hosts, each and every FPGA in the datacenter can reach every other one (at a scale of hundreds of thousands) in a small number of microseconds, without any intervening software. This capability allows hosts to use remote FPGAs for acceleration with low latency, improving the economics of the accelerator deployment, as hosts running services that do not use their local FPGAs can donate them to a global pool and extract value which would otherwise be stranded. Moreover, this design choice essentially turns the distributed FPGA resources into an independent computer in the datacenter, at the same scale as the servers, that physically shares the network wires with software. Figure 1a shows a logical view of this plane of computation.

This model offers significant flexibility. From the local perspective, the FPGA is used as a compute or a network accelerator. From the global perspective, the FPGAs can be managed as a large-scale pool of resources, with acceleration services mapped to remote FPGA resources. Ideally, servers not using all of their local FPGA resources can donate those resources to the global pool, while servers that need additional resources can request the available resources on remote servers. Failing nodes are removed from the pool with replacements quickly added. As demand for a service grows or shrinks, a global manager grows or shrinks the pools correspondingly. Services are thus freed from having a fixed ratio of CPU cores per FPGAs, and can instead allocate (or purchase, in the case of IaaS) only the resources of each type needed.

Space limitations prevent a complete description of the management policies and mechanisms for the global resource manager. Instead, this paper focuses first on the hardware architecture necessary to treat remote FPGAs as available resources for global acceleration pools. We describe the communication protocols and mechanisms that allow nodes in a remote acceleration service to connect, including a protocol called LTL (Lightweight Transport Layer) that supports lightweight connections between pairs of FPGAs, with mostly lossless transport and extremely low latency (small numbers of microseconds). This protocol makes the datacenter-scale remote FPGA resources appear closer than either a single local SSD access or the time to get through the host's networking stack. Then, we describe an evaluation system of 5,760 servers which we built and deployed as a precursor to hyperscale production deployment. We measure the performance characteristics of the system, using web search and network flow encryption as examples. We show that significant gains in efficiency are possible, and that this new architecture enables a much broader and more robust architecture for the acceleration

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

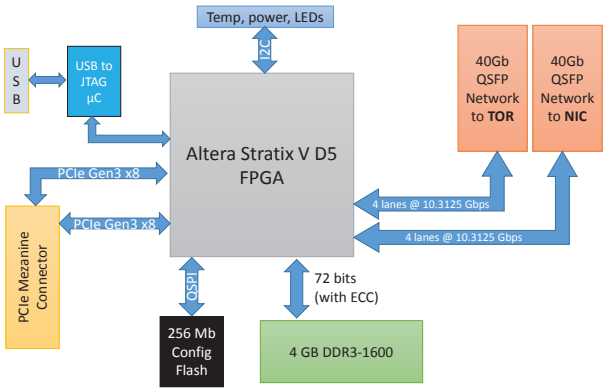

Fig. 2. Block diagram of the major components of the accelerator board.

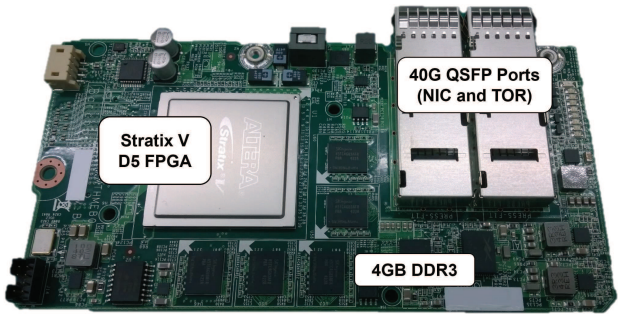

Fig. 3. Photograph of the manufactured board. The DDR channel is implemented using discrete components. PCIe connectivity goes through a mezzanine connector on the bottom side of the board (not shown).

of hyperscale datacenter services.

II. Hardware Architecture

There are many constraints on the design of hardware accelerators for the datacenter. Datacenter accelerators must be highly manageable, which means having few variations or versions. The environments must be largely homogeneous, which means that the accelerators must provide value across a plurality of the servers using it. Given services' rate of change and diversity in the datacenter, this requirement means that a single design must provide positive value across an extremely large, homogeneous deployment.

The solution to addressing the competing demands of homogeneity and specialization is to develop accelerator architectures which are programmable, such as FPGAs and GPUs. These programmable architectures allow for hardware homogeneity while allowing fungibility via software for different services. They must be highly *flexible* at the system level, in addition to being programmable, to justify deployment across a hyperscale infrastructure. The acceleration system we describe is sufficiently flexible to cover three scenarios: local compute acceleration (through PCIe), network acceleration, and global application acceleration, through configuration as pools of remotely accessible FPGAs. Local acceleration handles high-value scenarios such as search ranking acceleration where every server can benefit from having its own FPGA. Network acceleration can support services such as intrusion detection, deep packet inspection and network encryption which are critical to IaaS (e.g. "rental" of cloud servers), and which have such a huge diversity of customers that it makes it difficult to justify local compute acceleration alone economically. Global acceleration permits accelerators unused by their host servers to be made available for large-scale applications, such as machine learning. This decoupling of a 1:1 ratio of servers to FPGAs is essential for breaking the "chicken and egg" problem where accelerators cannot be added until enough applications need them, but applications will not rely upon the accelerators until they are present in the infrastructure. By decoupling the servers and FPGAs, software services that demand more FPGA capacity can harness spare FPGAs from other services that are slower to adopt (or do not require) the accelerator fabric.

In addition to architectural requirements that provide sufficient flexibility to justify scale production deployment, there are also physical restrictions in current infrastructures that must be overcome. These restrictions include strict power limits, a small physical space in which to fit, resilience to hardware failures, and tolerance to high temperatures. For example, the accelerator architecture we describe is the widely-used OpenCompute server that constrained power to 35W, the physical size to roughly a half-height half-length PCIe expansion card (80mm x 140 mm), and tolerance to an inlet air temperature of 70°C at 160 lfm airflow. These constraints make deployment of current GPUs impractical except in special HPC SKUs, so we selected FPGAs as the accelerator.

We designed the accelerator board as a standalone FPGA board that is added to the PCIe expansion slot in a production server SKU. Figure 2 shows a schematic of the board, and Figure 3 shows a photograph of the board with major components labeled. The FPGA is an Altera Stratix V D5, with 172.6K ALMs of programmable logic. The FPGA has one 4 GB DDR3-1600 DRAM channel, two independent PCIe Gen 3 x8 connections for an aggregate total of 16 GB/s in each direction between the CPU and FPGA, and two independent 40 Gb Ethernet interfaces with standard QSFP+ connectors. A 256 Mb Flash chip holds the known-good *golden image* for the FPGA that is loaded on power on, as well as one application image.

To measure the power consumption limits of the entire FPGA card (including DRAM, I/O channels, and PCIe), we developed a power virus that exercises nearly all of the FPGA's interfaces, logic, and DSP blocks—while running the card in a thermal chamber operating in worst-case conditions (peak ambient temperature, high CPU load, and minimum airflow due to a failed fan). Under these conditions, the card consumes 29.2W of power, which is well within the 32W TDP limits for a card running in a single server in our datacenter, and below the max electrical power draw limit of 35W.

The dual 40 Gb Ethernet interfaces on the board could allow for a private FPGA network as was done in our previous

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

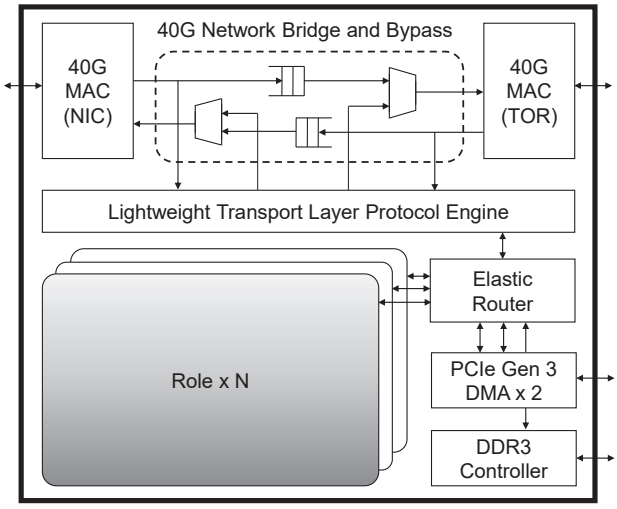

Fig. 4. The Shell Architecture in a Single FPGA.

	ALMs	MHz
Role	55340 (32%)	175
40G MAC/PHY (TOR)	9785 (6%)	313
40G MAC/PHY (NIC)	13122 (8%)	313
Network Bridge / Bypass	4685 (3%)	313
DDR3 Memory Controller	13225 (8%)	200
Elastic Router	3449 (2%)	156
LTL Protocol Engine	11839 (7%)	156
LTL Packet Switch	4815 (3%)	-
PCIe DMA Engine	6817 (4%)	250
Other	8273 (5%)	-
Total Area Used	**131350 (76%)**	-
Total Area Available	172600	-

Fig. 5. Area and frequency breakdown of production-deployed image with remote acceleration support.

work [4], but this configuration also allows the FPGA to be wired as a "bump-in-the-wire", sitting between the network interface card (NIC) and the top-of-rack switch (ToR). Rather than cabling the standard NIC directly to the ToR, the NIC is cabled to one port of the FPGA, and the other FPGA port is cabled to the ToR, as we previously showed in Figure 1b.

Maintaining the discrete NIC in the system enables us to leverage all of the existing network offload and packet transport functionality hardened into the NIC. This simplifies the minimum FPGA code required to deploy the FPGAs to simple bypass logic. In addition, both FPGA resources and PCIe bandwidth are preserved for acceleration functionality, rather than being spent on implementing the NIC in soft logic.

Unlike [5], both the FPGA and NIC have separate connections to the host CPU via PCIe. This allows each to operate independently at maximum bandwidth when the FPGA is being used strictly as a local compute accelerator (with the FPGA simply doing network bypass). This path also makes it possible to use custom networking protocols that bypass the NIC entirely when desired.

One potential drawback to the bump-in-the-wire architecture is that an FPGA failure, such as loading a buggy application, could cut off network traffic to the server, rendering the server unreachable. However, unlike a torus or mesh network, failures in the bump-in-the-wire architecture do not degrade any neighboring FPGAs, making the overall system more resilient to failures. In addition, most datacenter servers (including ours) have a side-channel management path that exists to power servers on and off. By policy, the known-good golden image that loads on power up is rarely (if ever) overwritten, so power cycling the server through the management port will bring the FPGA back into a good configuration, making the server reachable via the network once again.

A. Shell architecture

Within each FPGA, we use the partitioning and terminology we defined in prior work [4] to separate the application logic (Role) from the common I/O and board-specific logic (Shell) used by accelerated services. Figure 4 gives an overview of this architecture's major shell components, focusing on the network. In addition to the Ethernet MACs and PHYs, there is an intra-FPGA message router called the Elastic Router (ER) with virtual channel support for allowing multiple Roles access to the network, and a Lightweight Transport Layer (LTL) engine used for enabling inter-FPGA communication. Both are described in detail in Section V.

The FPGA's location as a bump-in-the-wire between the network switch and host means that it must always pass packets between the two network interfaces that it controls. The shell implements a bridge to enable this functionality, shown at the top of Figure 4. The shell provides a tap for FPGA roles to inject, inspect, and alter the network traffic as needed, such as when encrypting network flows, which we describe in Section III.

Full FPGA reconfiguration briefly brings down this network link, but in most cases applications are robust to brief network outages. When network traffic cannot be paused even briefly, partial reconfiguration permits packets to be passed through even during reconfiguration of the role.

Figure 5 shows the area and clock frequency of the shell IP components used in the production-deployed image. In total, the design uses 44% of the FPGA to support all shell functions and the necessary IP blocks to enable access to remote pools of FPGAs (i.e., LTL and the Elastic Router). While a significant fraction of the FPGA is consumed by a few major shell components (especially the 40G PHY/MACs at 14% and the DDR3 memory controller at 8%), enough space is left for the role(s) to provide large speedups for key services, as we show in Section III. Large shell components that are stable for the long term are excellent candidates for hardening in future generations of datacenter-optimized FPGAs.

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

B. Datacenter Deployment

To evaluate the system architecture and performance at scale, we manufactured and deployed 5,760 servers containing this accelerator architecture and placed it into a production datacenter. All machines were configured with the shell described above. The servers and FPGAs were stress tested using the power virus workload on the FPGA and a standard burn-in test for the server under real datacenter environmental conditions. The servers all passed, and were approved for production use in the datacenter.

We brought up a production Bing web search ranking service on the servers, with 3,081 of these machines using the FPGA for local compute acceleration, and the rest used for other functions associated with web search. We mirrored live traffic to the bed for one month, and monitored the health and stability of the systems as well as the correctness of the ranking service. After one month, two FPGAs had hard failures, one with a persistently high rate of single event upset (SEU) errors in the configuration logic, and the other with an unstable 40 Gb network link to the NIC. A third failure of the 40 Gb link to the TOR was found not to be an FPGA failure, and was resolved by replacing a network cable. Given aggregate datacenter failure rates, we deemed the FPGA-related hardware failures to be acceptably low for production.

We also measured a low number of soft errors, which were all correctable. Five machines failed to train to the full Gen3 x8 speeds on the secondary PCIe link. There were eight total DRAM calibration failures which were repaired by reconfiguring the FPGA. The errors have since been traced to a logical error in the DRAM interface rather than a hard failure. Our shell scrubs the configuration state for soft errors and reports any flipped bits. We measured an average rate of one bit-flip in the configuration logic every 1025 machine days. While the scrubbing logic often catches the flips before functional failures occur, at least in one case there was a role hang that was likely attributable to an SEU event. Since the scrubbing logic completes roughly every 30 seconds, our system recovers from hung roles automatically, and we use ECC and other data integrity checks on critical interfaces, the exposure of the ranking service to SEU events is low. Overall, the hardware and interface stability of the system was deemed suitable for scale production.

In the next sections, we show how this board/shell combination can support local application acceleration while simultaneously routing all of the server's incoming and outgoing network traffic. Following that, we show network acceleration, and then acceleration of remote services.

III. Local Acceleration

As we described earlier, it is important for an at-scale datacenter accelerator to enhance local applications *and* infrastructure functions for different domains (e.g. web search and IaaS). In this section we measure the performance of our system on a large datacenter workload.

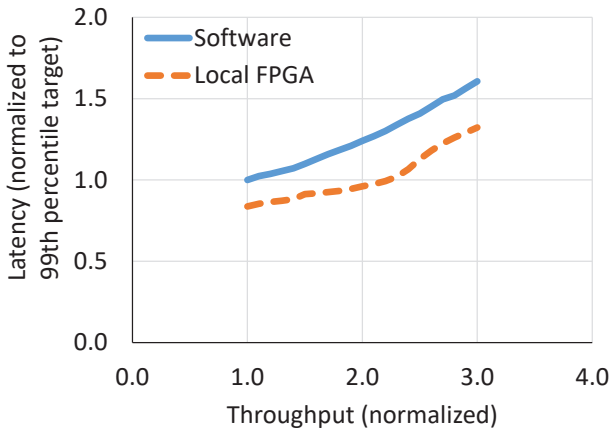

Fig. 6. 99% Latency versus Throughput of ranking service queries running on a single server, with and without FPGAs enabled.

A. Bing Search Page Ranking Acceleration

We describe Bing web search ranking acceleration as an example of using the local FPGA to accelerate a large-scale service. This example is useful both because Bing search is a large datacenter workload, and since we had described its acceleration in depth on the Catapult v1 platform [4]. At a high level, most web search ranking algorithms behave similarly; query-specific features are generated from documents, processed, and then passed to a machine learned model to determine how relevant the document is to the query.

Unlike in [4], we implement only a subset of the feature calculations (typically the most expensive ones), and neither compute post-processed synthetic features nor run the machine-learning portion of search ranking on the FPGAs. We do implement two classes of features on the FPGA. The first is the traditional finite state machines used in many search engines (e.g. "count the number of occurrences of query term two"). The second is a proprietary set of features generated by a complex dynamic programming engine.

We implemented the selected features in a *Feature Functional Unit* (FFU), and the *Dynamic Programming Features* in a separate DPF unit. Both the FFU and DPF units were built into a shell that also had support for execution using remote accelerators, namely the ER and LTL blocks as described in Section V. This FPGA image also, of course, includes the network bridge for NIC-TOR communication, so all the server's network traffic is passing through the FPGA while it is simultaneously accelerating document ranking. The pass-through traffic and the search ranking acceleration have no performance interaction.

We present results in a format similar to the Catapult results to make direct comparisons simpler. We are running this image on a full production bed consisting of thousands of servers. In a production environment, it is infeasible to simulate many different points of query load as there is substantial infrastructure upstream that only produces requests at the rate of arrivals. To produce a smooth distribution with repeatable results, we used a single-box test with a stream

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.