Fully-Buffered DIMM Memory Architectures: Understanding Mechanisms, Overheads and Scaling

Brinda Ganesh[†], Aamer Jaleel[‡], David Wang[†], and Bruce Jacob[†]

[†]University of Maryland, College Park, MD

[‡]VSSAD, Intel Corporation, Hudson, MA

{brinda, blj}@eng.umd.edu

Abstract

Performance gains in memory have traditionally been obtained by increasing memory bus widths and speeds. The diminishing returns of such techniques have led to the proposal of an alternate architecture, the Fully-Buffered DIMM. This new standard replaces the conventional memory bus with a narrow, high-speed interface between the memory controller and the DIMMs. This paper examines how traditional DDRx based memory controller policies for scheduling and row buffer management perform on a Fully-Buffered DIMM memory architecture. The split-bus architecture used by FBDIMM systems results in an average improvement of 7% in latency and 10% in bandwidth at higher utilizations. On the other hand, at lower utilizations, the increased cost of serialization resulted in a degradation in latency and bandwidth of 25% and 10% respectively. The split-bus architecture also makes the system performance sensitive to the ratio of read and write traffic in the workload. In larger configurations, we found that the FBDIMM system performance was more sensitive to usage of the FBDIMM links than to DRAM bank availability. In general, FBDIMM performance is similar to that of DDRx systems, and provides better performance characteristics at higher utilization, making it a relatively inexpensive mechanism for scaling capacity at higher bandwidth requirements. The mechanism is also largely insensitive to scheduling policies, provided certain ground rules are obeyed.

1. Introduction

The growing size of application working sets, the popularity of software-based multimedia and graphics workloads, and increased use of speculative techniques, have contributed to the rise in bandwidth and capacity requirements of computer memory sub-systems. Capacity needs have been met by building increasingly dense DRAM chips (the 2Gb DRAM chip is expected to hit the market in 2007) and bandwidth needs have been met by scaling front-side bus data rates and using wide, parallel buses. However, the traditional bus organization has reached a point where it fails to scale well into the future.

The electrical constraints of high-speed parallel buses complicate bus scaling in terms of loads, speeds, and widths [1]. Consequently the maximum number of DIMMs per channel in successive DRAM generations has been decreasing. SDRAM channels have supported up to 8 DIMMs of memory, some types of DDR channels support only 4 DIMMs, DDR2 channels have but 2 DIMMs, and DDR3 channels are expected to support only a single DIMM. In addition the serpentine routing required for electrical path-length matching of the data wires becomes challenging as bus widths increase. For the bus widths of today, the motherboard area occupied by a single channel is significant, complicating the task of adding capacity by increasing the number of channels. To address these scalability issues, an alternate DRAM technology, the Fully Buffered Dual Inline Memory Module (FBDIMM) [2] has been introduced.

The FBDIMM memory architecture replaces the shared parallel interface between the memory controller and DRAM chips with a point-to-point serial interface between the memory controller and an intermediate buffer, the Advanced Memory Buffer (AMB). The on-DIMM interface between the AMB and the DRAM modules is identical to that seen in DDR2 or DDR3 systems, which we shall refer to as DDRx systems. The serial interface is split into two uni-directional buses, one for read traffic (northbound channel) and another for write and command traffic (southbound channel), as shown in Fig 1. FBDIMMs adopts a packet-based protocol that bundles commands and data into frames that are transmitted on the channel and then converted to the DDRx protocol by the AMB.

The FBDIMMs' unique interface raises questions on whether existing memory controller policies will continue to be relevant to future memory architectures. In particular we ask the following questions:

- How do DDRx and FBDIMM systems compare with respect to latency and bandwidth?
- What is the most important factor that impacts the performance of FBDIMM systems?
- How do FBDIMM systems respond to choices made for parameters like row buffer management policy, scheduling policy and topology?

Our study indicates that an FBDIMM system provides significant capacity increases over a comparable DDRx system at relatively little cost. for systems with high bandwidth requirements. However, at lower utilization requirements, the FBDIMM protocol results in an average latency degradation of 25%. We found the extra cost of serialization is also apparent in configurations with shallow channels i.e. 1-2 ranks per channel, but can be successfully hidden in systems with deeper channels by taking advantage of the available DIMM-level parallelism.

The sharing of write data and command bandwidth is a significant bottleneck in these systems. An average of 20-30% of the southbound channel bandwidth is wasted due to the inability of the memory controller to fully utilize the available command slots in a southbound frame. These factors together contribute to the queueing delays for a read transaction in the system.

More interestingly, we found that the general performance characteristics and trends seen for applications executing on the FBDIMM architecture were similar in many ways to that seen in DDRx systems. We found that many of the memory controller policies that were most effective in DDRx systems were also effective in an FBDIMM system.

This paper is organized as follows. Section 2 describes the FBDIMM memory architecture and protocol. Section 3 describes the related work. Section 4 describes the experimental framework and methodology. Section 5 has detailed results and analysis of the observed behavior. Section 6 summarizes the final conclusions of the paper.

2. Background

2.1 FBDIMM Overview

As DRAM technology has advanced, channel speed has improved at the expense of channel capacity; consequently channel capacity has dropped from 8 DIMMs to a single DIMM per channel. This is a significant limitation—for a server designer, it is critical, as servers typically depend on memory capacity for their performance. There is an obvious dilemma: future designers would like both increased channel capacity and increased data rates—but how can one provide both?

For every DRAM generation, graphics cards use the same DRAM technology as is found in commodity DIMMs, but operate their DRAMs at significantly higher speeds by avoiding multi-drop connections. The trend is clear: to improve bandwidth, one must reduce the number of impedance discontinuities on the bus.

This is achieved in FBDIMM system by replacing the multidrop bus with point-to-point connections. Fig 1 shows the altered memory interface with a narrow, high-speed channel connecting the master memory controller to the DIMM-level memory controllers (called *Advanced Memory Buffers*, or *AMBs*). Since each DIMM-to-DIMM connection is a point-to-point connection, a channel becomes a *de facto* multi-hop store & forward network. The FBDIMM architecture limits the channel length to 8 DIMMs, and the narrower inter-module bus requires roughly one third as many pins as a traditional organization. As a result, an FBDIMM organization can handle roughly *24 times* the storage capacity of a single-DIMM DDR3-based system, without sacrificing any bandwidth and even leaving headroom for increased intra-module bandwidth.

The AMB acts like a pass-through switch, directly forwarding the requests it receives from the controller to successive DIMMs and forwarding frames from southerly DIMMs to northerly DIMMs or the memory controller. All frames are processed to determine whether the data and commands are for the local DIMM.

2.2 FBDIMM protocol

The FBDIMM system uses a serial packet-based protocol to communicate between the memory controller and the DIMMs. Frames may contain data and/or commands. Commands include DRAM commands such as row activate (RAS), column read (CAS), refresh (REF) and so on, as well as channel commands such as write to configuration registers, synchronization commands etc. Frame scheduling is performed exclusively by the memory controller. The AMB only converts the serial protocol to DDRx based commands without implementing any scheduling functionality.

The AMB is connected to the memory controller and/or adjacent DIMMs via serial uni-directional links: the southbound channel which transmits both data and commands and the northbound channel which transmits data and status information. The southbound and northbound data paths are respectively 10 bits and 14 bits wide respectively. The channel is dual-edge clocked i.e. data is transmitted on both clock edges. The FBDIMM channel clock operates at 6 times the speed of the DIMM clock i.e. the link speed is 4 Gbps for a 667 Mbps DDRx system. Frames on the north and south bound channel require 12 transfers (6 FBDIMM channel clock cycles) for transmission. This 6:1 ratio ensures that the FBDIMM frame rate matches the DRAM command clock rate.

Southbound frames comprise both data and commands and are 120 bits long; data-only northbound frames are 168 bits long. In addition to the data and command information, the frames also carry header information and a frame CRC checksum that is used to check for transmission errors.

The types of frames defined by the protocol [3] are illustrated in the figure 2. They are

Command Frame: comprises up to three commands (Fig 2(a)). Each command in the frame is sent to a separate

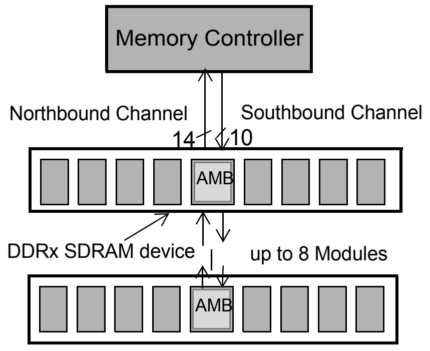

Figure 1. FBDIMM Memory System. TIn the FBDIMM organization, there are no multi-drop busses; DIMM-to-DIMM connections are point-to-point. The memory controller is connected to the nearest AMB, via two uni-directional links. The AMB is in turn connected to its southern neighbor via the same two links.

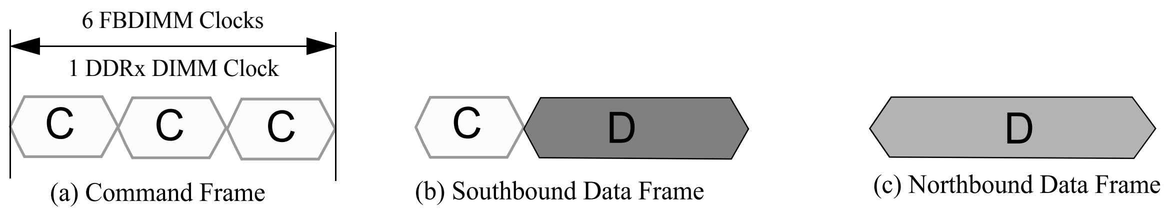

Figure 2. FBDIMM Frames. The figure illustrates the various types of FBDIMM frames used for data and command transmission.

DIMM on the southbound channel. In the absence of available commands to occupy a complete frame, no-ops are used to pad the frame.

Command + Write Data Frame: carries 72 bits of write data - 8 bytes of data and 1 byte of ECC - and one command on the southbound channel (Fig 2b). Multiple such frames are required to transmit an entire data block to the DIMM. The AMB of the addressed DIMM buffers the write data as required.

Data frame: comprises 144 bits of data or 16 bytes of data and 2 bytes of ECC information (Fig 2(c)). Each frame is filled by the data transferred from the DRAM in two beats or a single DIMM clock cycle. This is a northbound frame and is used to transfer read data.

A northbound data frame transports 18 bytes of data in 6 FBDIMM clocks or 1 DIMM clock. A DDRx system can burst back the same amount of data to the memory controller in two successive beats lasting an entire DRAM clock cycle. Thus, the read bandwidth of an FBDIMM system is the same as that of a single channel of DDRx system. A southbound frame on the other hand transports half the amount of data, 9 bytes as compared to 18 bytes, in the same duration. Thus, the FBDIMM southbound channel transports half the number of bytes that a DDRx channel can burst write in 1 DIMM clock, making the write bandwidth in FBDIMM systems one half that available in a DDRx system. This makes the total bandwidth available in an FBDIMM system 1.5 times that in a DDRx system.

The FBDIMM has two operational modes: the fixed latency mode and the variable latency mode. In fixed latency mode, the round-trip latency of frames is set to that of the furthest DIMM. Hence, systems with deeper channels have larger average latencies. In the variable latency mode, the round-trip latency from each DIMM is dependent on its distance from the memory controller.

Figure 3 shows the processing of a read transaction in an FBDIMM system. Initially a command frame is used to transmit a command that will perform row activation. The AMB translates the request and relays it to the DIMM. The memory controller schedules the CAS command in a following frame. The AMB relays the CAS command to the DRAM devices which burst the data back to the AMB. The AMB bundles two consecutive bursts of a data into a single northbound frame and transmits it to the memory controller. In this example, we assume a burst length of 4 corresponding to 2 FBDIMM data frames. Note that although the figures do not identify parameters like t_CAS, t_RCD and t_CWD [4, 5] the memory controller must ensure that these constraints are met.

Figure 4 shows the processing of a memory write transaction in an FBDIMM system. The write data requires twice the number of FBDIMM frames than the read data. All read and write data are buffered in the AMB before being relayed on, indicated by the staggered timing of data on the respective buses. The CAS command is transmitted in the same frame as the last chunk of data.

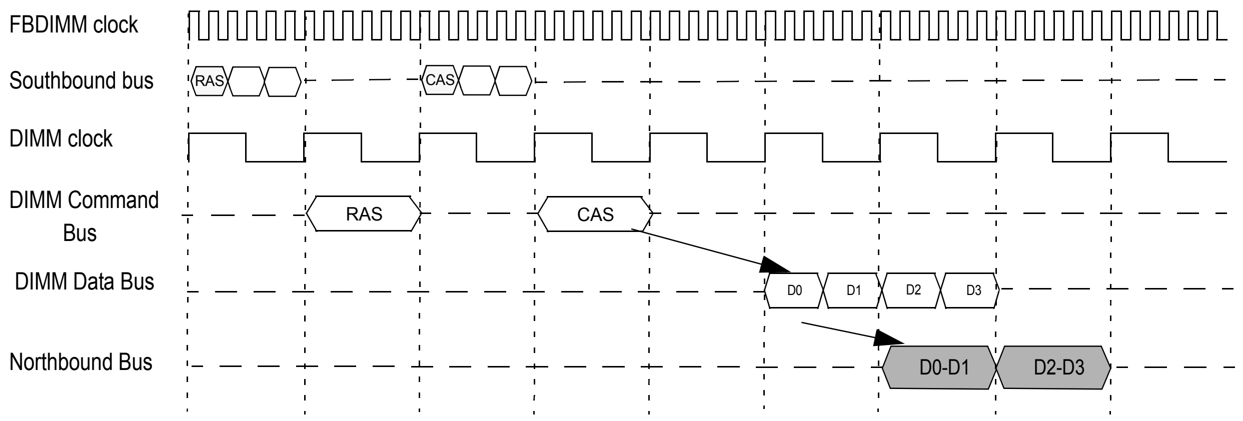

Figure 3. Read Transaction in FBDIMM system. The figure shows how a read transaction is performed in an FBDIMM system. The FBDIMM serial buses are clocked at 6 times the DIMM buses. Each FBDIMM frame on the southbound bus takes 6 FBDIMM clock periods to transmit. On the northbound bus a frame comprises of two DDRx data bursts.

3. Related Work

Burger et. al. [6] demonstrated that many high-performance mechanisms lower latency by increasing bandwidth demands. Cuppu et. al. [7] demonstrated that memory manufacturers have successfully scaled data-rates of DRAM chips by employing pipelining but have not reduced the latency of DRAM device operations. In their follow-on paper [8] they showed that system bus configuration choices significantly impact overall system performance.

There have been several studies of memory controller polices that examine how to lower latency while simultaneously improving bandwidth utilization. Rixner et. al. [9] studied how re-ordering accesses at the controller to increase row buffer hits can be used to lower latency and improve bandwidth for media processing streams. Rixner[10] studied how such re-ordering benefitted from the presence of SRAM caches on the DRAM for web servers. Natarajan et. al. [11] studied how memory controller policies for row buffer management and command scheduling impact latency and sustained bandwidth.

Lin et. al. [12] studied how memory controller based prefetching can lower the system latency. Zhu et. al. [13] studied how awareness of resource usage of threads in an SMT could be used to prioritize memory requests. Zhu et. al. [14] study how split-transaction support would lower the latency of individual operations.

Intelligent static address mapping techniques have been used by Lin et. al. [12] and Zhang et. al. [15] to lower memory latency by increasing row-buffer hits. The Impulse group at University of Utah [16] proposed adding an additional layer of address mapping in the memory controller to reduce memory latency by mapping non-adjacent data to the same cacheline and thus increasing cacheline sub-block usage.

This paper studies how various memory controller policies perform on the fully-buffered DIMM system. By doing a detailed analysis of the performance characteristics of DDRx specific scheduling policies and row buffer management policies on the FBDIMM memory system, the bottlenecks in these systems and their origin are identified. To the best of our knowledge, this is the first study to examine the FBDIMM in this detail.

4. Experimental Framework

This study uses DRAMsim [4], a stand-alone memory system simulator. DRAMsim provides a detailed execution-driven model of a fully-buffered DIMM memory system. The simulator also supports the variation of memory system parameters of interest including scheduling policies, memory configuration i.e. number of ranks and channels, address mapping policy etc.

We studied four different DRAM types: a conventional DDR2 system with a data-rate of 667Mbps, a DDR3 system with a data rate of 1333Mbps and their corresponding FBDIMM systems: an FBDIMM organization with DDR2 on the DIMM and another with DDR3 on the DIMM. FBDIMM modules are modelled using the parameters available for a 4GB module [3]; DDR2 device are modelled as 1Gb parts with 8 banks of memory [5]; DDR3 parameters were based on the proposed JEDEC standard[17]. A combination of multi-program and multi-threaded workloads are used. We used application memory traces as inputs.

- **SVM-RFE (Support Vector Machines Recursive Feature Elimination)** [18] is used to eliminate gene redundancy from a given input data set to provide compact gene subsets. This is a read intensive workload (reads comprise approximately 90% of the traffic). It is part of the BioParallel suite[24].
- **SPEC-MIXED** comprising of 16 independent SPEC [19] applications including AMMP, applu, art, gcc, lucas, mgrid, swim, vortex, wupwise, bzip2, galgel, gzip, mcf, parser, twolf and vpr. The combined workload has a read to write traffic ratio of approximately 2:1.
- **UA** is Unstructured Adaptive (UA) benchmark which is part of the NAS benchmark suite, NPB 3.1 OMP [20]. This workload has a read to write traffic ratio of approximately 3:2.
- **ART** the Open MP version of the SPEC 2000 ART benchmark which forms part of the SPEC OMP 2001 suite[21]. This has a read to write ratio of around 2:1.

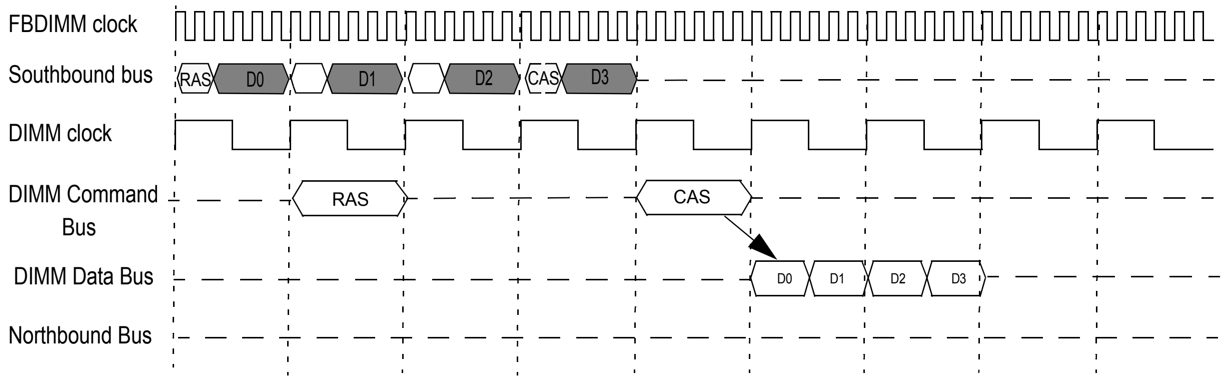

Figure 4. Write Transaction in FBDIMM system. The figure shows how a write transaction is performed in an FBDIMM system. The FBDIMM serial buses are clocked at 6 times the DIMM buses. Each FBDIMM frame on the southbound bus takes 6 FBDIMM clock periods to transmit. A 32 byte cacheline takes 8 frames to be transmitted to the DIMM.

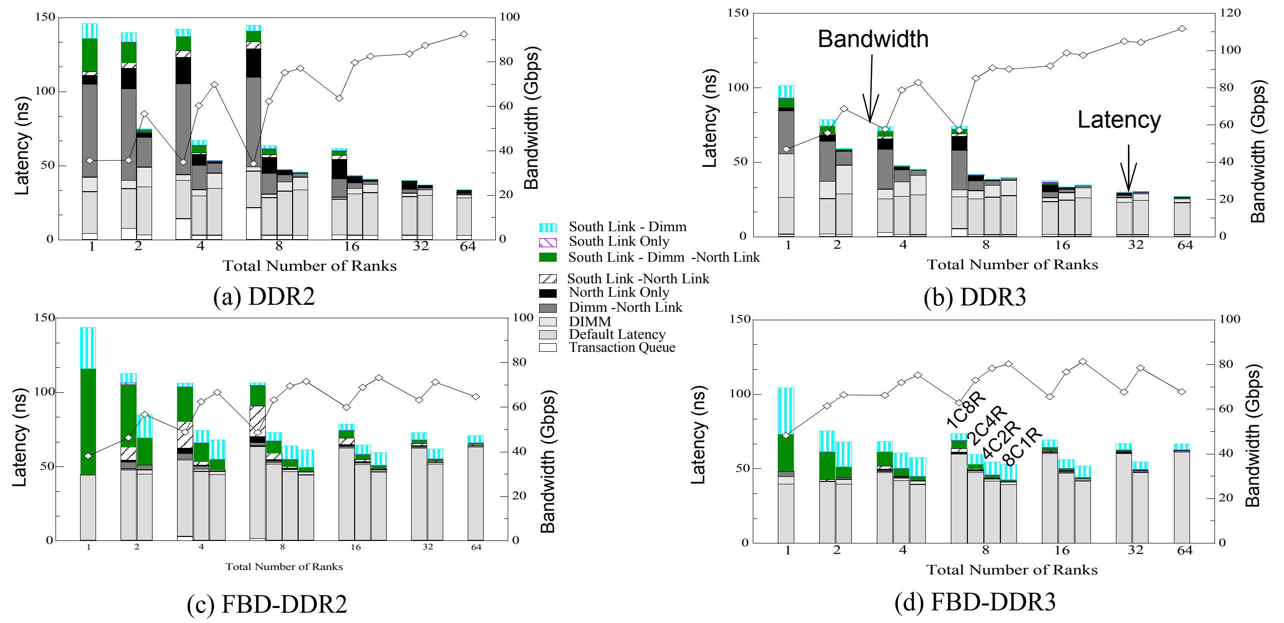

Figure 5. Latency and Bandwidth characteristics for different DRAM technologies. The figure shows the best latency and bandwidth characteristics for the SPEC-Mixed workload in an open page system. Systems with identical numbers of dimms are grouped together on the x-axis. Within a group, bars on the left represent topologies with lesser numbers of channels. The left y-axis depicts latency in nano-seconds, while the right y-axis plots bandwidth in Gbps. Note that the lines represent bandwidth and the bars represent latency.

All multi-threaded traces were gathered using CMP$im [22], a tool that models the cache hierarchy. Traces were gathered using 16-way CMP processor models with 8 MB of last-level cache. SPEC benchmark traces were gathered using Alpha 21264 simulator sim-alpha [23], with a 1 MB last-level cache and an in-order core.

Several configuration parameters that were varied in this study include:

Number of ranks and their configuration. FBDIMM systems support six channels of memory, each with up to eight DIMMs. This study expands that slightly, investigating a configuration space of 1-64 ranks, with ranks organized in 1-8 channels of 1-8 DIMMs each. Though many of these configurations are not relevant for DDRx systems, we study them the impact of serialization on performance.We study configurations with one rank per DIMM.

Row-buffer management policies . We study both open-page and closed-page systems. Note that the address mapping policy is selected so as to reduce bank conflicts in the system. The address mapping policies sdram_close_page_map and sdram_hiperf_map [4] provide the required mapping for closed page and open page systems respectively.

Scheduling policies . Several DDRx DRAM command scheduling policies were studied. These include

- **Greedy** The memory controller issues a command as soon as it is ready to go. Priority is given to DRAM commands associated with older transactions.
- **Open Bank First (OBF)** is used only for open-page systems. Here priority is given to all transactions which are issued to a currently open bank.
- **Most pending** is the "most pending" scheduling policy proposed by Rixner [9], where priority is given to DRAM commands to banks which have the maximum number of transactions.
- **Least pending:** is the "least pending" scheduling policy proposed by Rixner [9]. Commands to transactions to banks with the least number of outstanding transactions are given priority.
- **Read-Instruction-Fetch-First (RIFF):** prioritizes memory read transactions (data and instruction fetch) over memory write transactions.

5. Results

We study two critical aspects of the memory systems performance: the average latency of a memory read operation, and the sustained bandwidth.

5.1 The Bottom Line

Figure 5 compares the behavior of the different DRAM systems studied. The graphs show the variation in latency and bandwidth with system configuration for the SPEC-Mixed workload executing in an open page system. In the graphs systems with different topologies but identical numbers of ranks are grouped together on the x-axis. Within a group, the bars on the left represent topologies with more ranks/channel and as one moves from left to right, the same ranks are distributed across more channels. For example, for the group of configurations with a total of 8 ranks of memory, the

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.