Solid-State File Caching for Performance and Scalability

by Michael Casey
Solid Data Systems, Inc.

Contents

While Internet bandwidth issues are being resolved with broadband connections, a transaction-processing bottleneck is merging as a critical issue. A single e-mail, stock trade or book purchase requires very little bandwidth, but customer response time and system resiliency often suffer when application servers must process millions of transactions per day. Where a server is waiting on mechanical disk drives to read or write the transaction data, one solution is to distribute transactions across many servers and disks. However, if mechanical latency can be eliminated, existing servers can handle many more transactions—multiplying scalability while speeding response time. Where a small percentage of the data files consume most of the I/O activity, the solution is to place those files in a high-performance file cache. This approach typically uses solid-state disk (SSD) for file caching. By eliminating mechanical latency, SSD turbocharges performance for Internet and e-business transactions.

The Problem with Disk I/O

As they scale up applications such as e-mail, ERP, and CRM, many enterprises are finding that traditional cached disk arrays do not deliver the needed transaction rates and response times. This is inevitable since disk drive capacities are growing exponentially while disk I/O performance is not.

Figure 1 *below illustrates this divergence: disk drive capacities have increased by a factor of 30 times in the past eight years, while the random-access performance of an individual mechanical disk drive has improved only slightly. On a per-Gbyte basis, disk subsystem performance actually decreases with each improvement in disk storage density and cost.*

Figure 2 *below illustrates this trend in terms of access density, the I/Os per second (or IOPS) divided by the maximum capacity per disk drive. (Reference: "Access Density—Key to Disk Performance" by Randy Kerns, Storage, Inc., Q2 1999. http://www.wwpi.com)*

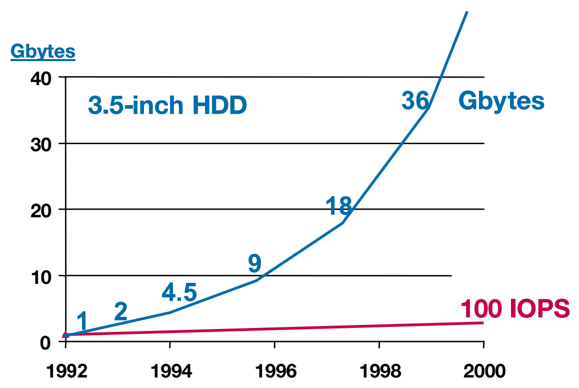


Figure 1. Technology Divergence: Mechanical Disk Drive Capacity vs. Performance

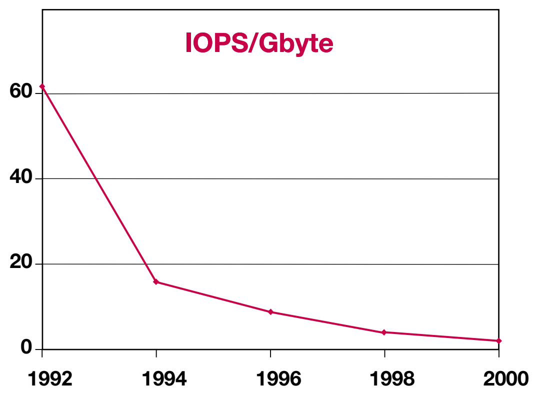


Figure 2. Disk Drive Access Density Trend

For example, a single-bay EMC disk array can hold 96 disk drives. Each 3.5-inch drive could hold 9 Gbytes when EMC moved from 5.25-inch to 3.5-inch drives about 3 years ago. Now each drive can hold 18 or 36 Gbytes, and Seagate is now shipping 50 Gbyte drives as well. However, each of the 96 drives still delivers only about 100 IOPS, so the raw back-end performance of the box is around 9,600 IOPS—no matter which capacity disk drives are used.

To improve performance on random-I/O workloads EMC and other disk array vendors incorporate semiconductor cache inside the disk subsystems. The performance benefit of such a cached RAID approach depends on the cache hit ratio, which in turn depends on the data access patterns of the applications being supported.

In many of today's transaction-intensive applications, enterprises and service providers are finding that they must spread the data over a large number of disk drives and array frames to achieve an adequate number of I/Os per Gbyte of data stored. Even then, mechanical disk latencies can significantly constrain performance and scalability: the application server may be wasting a significant portion of its potential processing power, while it waits for the disk subsystem to complete I/O requests.

Architectural Solutions for Disk I/O Performance Scaling

Today's performance issues with cached RAID solutions are part of a long-term trend, driven by the rapid increase in disk capacity. Over the past 10 years, as the capacity per drive has increased faster than the I/O performance per drive, system designers have adopted a series of architectural innovations to maintain application performance levels and to respond to demands for increased transaction throughput. Figure 3 illustrates this progression of architectural innovations.

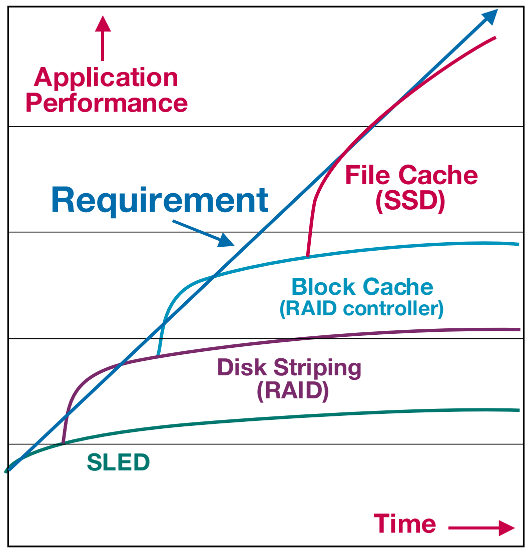


Figure 3 . Architectural Solutions for Disk I/O Performance and Scalability

The authors of the famous RAID papers (a group of computer scientists at UC Berkeley) described the problem with a SLED—a Single Large Expensive Disk—and proposed various RAID architectures as possible solutions to the performance problem (http://sunsite.berkeley.edu).

In essence, the idea was that system designers could improve performance by striping the data across an array of independent disk drives. A current example would be Sun's A5200 disk array, with data striping accomplished in server software. This data-striping architecture provided significant performance improvement in suitable applications.

Next, in response to the ever-increasing demands for application performance and scalability, RAID vendors began to design RAID controller hardware with some amount of “block cache” (discussed below in more detail). RAID controller cache still adequately addresses the I/O performance needs of many common applications.

However, cached RAID solutions are starting to run out of gas in high-performance, transaction-intensive applications: while new disk drive generations are decreasing the access density of the back-end disk arrays, the performance demands on e-business infrastructures are being driven to new levels by exploding demand, unpredictable peak loads, and an increasingly impatient population of on-line customers. It’s time for another architectural innovation.

As Figure 3 suggests, that next innovation is a move to “hot file” caching on external, persistent solid-state storage.

File Cache vs. Block Cache

For many transaction-intensive applications, it is possible to identify a small set of files that consume most of the I/O activity and place those files in a high-performance cache. This approach typically uses solid-state disk (SSD) for file caching and increases overall transaction performance by a factor of 200% or more on the existing servers. File caching differs from block caching in several respects.

RAID Cache

RAID cache—“block caching” in a RAID controller— is based on data blocks. Each block is identified by a SCSI block address (for example), and the RAID controller has no knowledge of which blocks are part of which file. It chooses what to cache (and what to flush from cache) based on historical usage statistics on the individual blocks (or disk tracks, in the case of EMC). The caching algorithm looks at the usage history, and tries to determine what will be needed next by the application. Since the controller cache is much smaller than the total amount of data stored on the disk drives, only a small percentage of the data can be kept in the cache. A given data block may reside in cache for only a few seconds or minutes, before it is flushed from cache.

The effectiveness of block caching depends on the application’s data access patterns, and usually an application will reach a point of diminishing returns—a point where adding more cache will not deliver much additional performance improvement. Once the application reaches that point, the next step is to start caching entire files in a very fast I/O device.

File Cache

File cache effectiveness depends on the user's understanding of the application structure—i.e., the identification of the application-specific "hot files." Once the hot files have been identified, selected files are moved to the file cache—as a policy decision, not a statistical extrapolation. The hot files may reside on the SSD for days, weeks or months. The "hit ratio" on data in the file cache is always 100%, since the entire file is always in the cache and available for access.

Alternative Cache Locations: A system or application architect can establish a file cache (or a block cache) in any of several locations. For example, Oracle buffer caches and Unix file system buffers generally make use of the system's main memory. This is a suitable location for read caching, but is not recommended for write caching due to the volatility of main memory.

Some computer hardware vendors do offer non-volatile cache inside the server, using DRAM with battery backup—Sun's fast write cache is one example—and that is a suitable approach for in-line block cache in single-server configurations of limited size. However, if the facility is to be used as a file cache and shared among multiple servers in a cluster or storage area network (SAN), then the preferred implementation is a separate, non-volatile file-caching device or appliance.

Several RAID subsystem vendors also offer a non-volatile file-caching facility, as an optional feature of the RAID controller cache. The most notable examples are the EMC Symmetrix, the Hitachi 7700E, and HP's XP256 disk array. In essence, the RAID vendor offers the ability to dedicate part of the cache for storage of specific files (or virtual volumes). EMC calls this feature "PermaCache" and HP calls it "HP SureStore E Cache LUN XP." This "SSD in RAID cache" approach may satisfy a need for a small amount of file cache, if an adequate number of spare cache slots are available within the disk array frame. However, it is not easily scalable beyond the cache size limit of the existing frame. Few enterprises will want to buy another Symmetrix or XP256 frame, just to install another one or two Gbytes of file cache. Also, in many cases, the RAID subsystem needs all the block cache it can get; allocating part of that limited resource to a file-caching facility will impact performance of the cached RAID facility.

In this context, rapidly-growing enterprises should avoid the mistake of settling on a monolithic architecture that ties cache capacity to disk-array frames, and thus cannot readily scale over a wide range of file cache capacities.

A more scalable architecture is provided by a separate file cache: it enables users to add file-caching resources independently from the RAID frames. By specifying an independent file-caching device, an enterprise or system integrator defines an architecture in which performance can be scaled more smoothly, independent of raw disk capacity or number of server and RAID frames.

When Does File Caching Make Sense?

Two conditions are necessary to make solid-state file caching a good bet:

(1) The application server must be I/O bound.
(2) The I/Os must be skewed: a small percentage of the files must drive a large percentage of the I/O activity.

Thus, when analyzing a transaction performance problem on an application server, the first step is to determine whether the server is I/O bound. If the server is compute-bound rather than I/O bound, then improving the I/O performance is unlikely to help. System administrators typically use tools such as SAR and V$SYSSTAT to determine whether or not the server is “waiting for I/O” a significant percentage of the time, under peak load conditions.

If the server is indeed I/O bound for a critical application at peak load, the next step is to determine whether the I/O activity is skewed. (If it isn’t, then the application’s needs can generally be met by adding more disk spindles in a cached RAID subsystem.) In some applications, answering this question is fairly easy, as the identities of the hot files are inherent in the design of the application.

For example, in many e-mail and messaging applications the message queues represent a high percentage of the total I/O, on a small percentage of the data. They are also very write-intensive files. This skewed I/O distribution is a feature of the application design, and thus the application is a good fit for architectural adoption of solid-state file caching. Figure 4 illustrates this concept: a small, I/O-intensive fraction of the data is moved from cached RAID to a separate, non-volatile file cache.

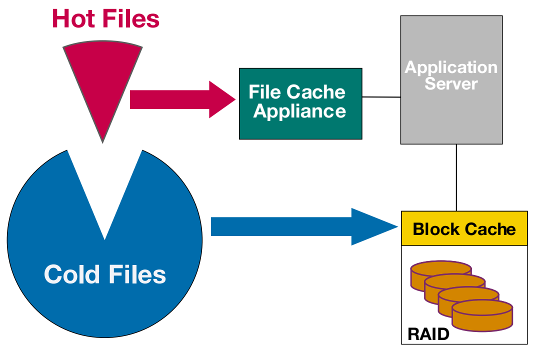


Figure 4. File Cache - a Scalable I/O Architecture

Performance Impact

In suitable applications, addition of a solid-state file cache to an existing server can boost throughput by a factor 4 or even 8. This enables the system administrator to deliver the required performance and service levels, without purchasing and managing several additional servers.

Figure 5 shows the impact of solid-state file caching on the performance and scalability of a Sendmail message server. In the original configuration—a Sun server with all data stored on mechanical disk drives—the server was in “wait for I/O” mode 40% of the time at peak load.

After the message queues were moved to a solid-state file cache, the I/O wait was largely eliminated, and the server was able to process four times as many messages per second. The internet service provider was able to scale the application by increasing the throughput of each server, rather than buying four times as many servers and managing a more complex hardware environment.

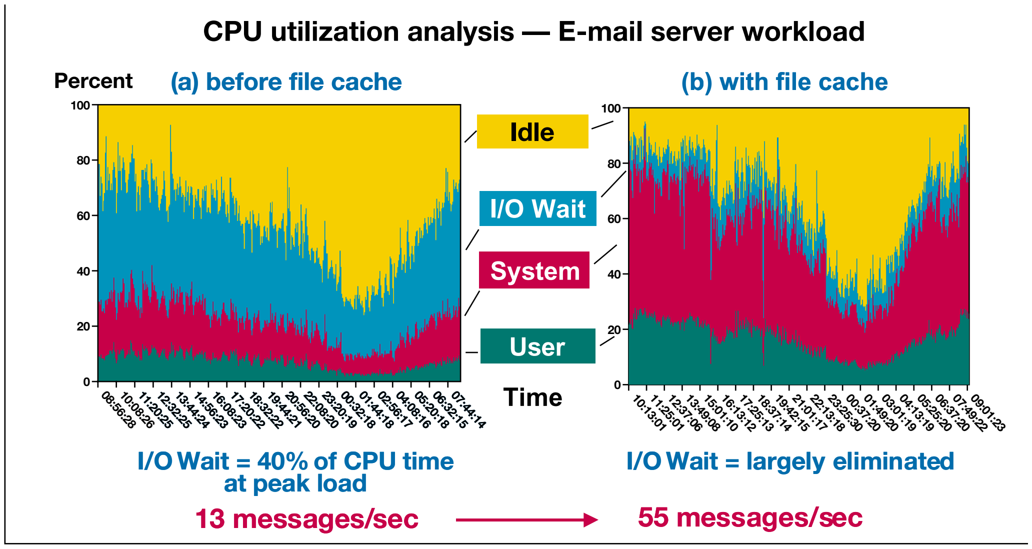


Figure 5. File Cache as Performance Multiplier

As this ISP's business continues to grow, the e-mail application-server configuration is simply replicated as part of a scalable infrastructure: one solid-state file-cache hardware module for each pair of new servers installed.

Oracle Databases

Oracle databases are often more complex than Sendmail servers, and more difficult to analyze. Oracle tends to spread disk I/Os across multiple devices, even within a single file; and operating-system tools do not readily reveal which files are causing most of the device I/O traffic. In general, we know that the most likely candidates for file caching in an Oracle database are very active index files, re-do logs and temp spaces. However, a large Oracle implementation might have multiple files of each type, so hot file identification tools and procedures are important parts to Database Administrator's or sysadmin's professional toolbox.

As in the previous example, the first step in I/O performance analysis is to determine whether the application server is I/O bound. SAR scripts and other tools can provide the basis for such an analysis. Various vendors offer services and tools to facilitate the process (e.g., see http://www.soliddata.com/products/iodynamics.html).

Figure 6 shows the CPU utilization analysis for a Peoplesoft application running on an Oracle database. In this example, the I/O Wait represented 63% of CPU time at peak load.

Based on the high percentage of wasted CPU cycles—time the processor spends waiting for disk I/O requests to complete, during peak-load conditions—this Peoplesoft application appears to be a good candidate for I/O improvement. However, at this point, we do not know for sure whether the solution is simply to add more disk spindles and I/O paths—or whether adding solid-state file cache would be the most effective solution.

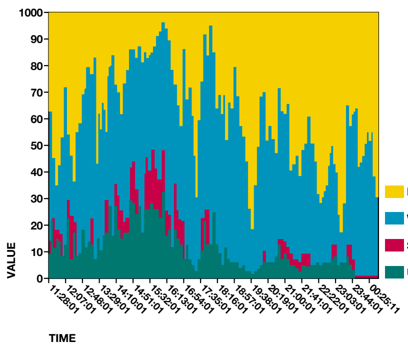


Figure 6. CPU Utilization Analysis for a Peoplesoft Application

In other words, we still must answer the second key question: is the disk I/O uniform or skewed? And in Oracle applications, simply looking at system-level device I/O statistics is not very useful, since Oracle tends to spread the hot files across many devices. Such a policy may be efficient in a world where mechanical disk drives are the only I/O devices available, but it ties the application performance to the average performance of the disk farm, and it complicates the task of moving high-activity files onto a faster device category.

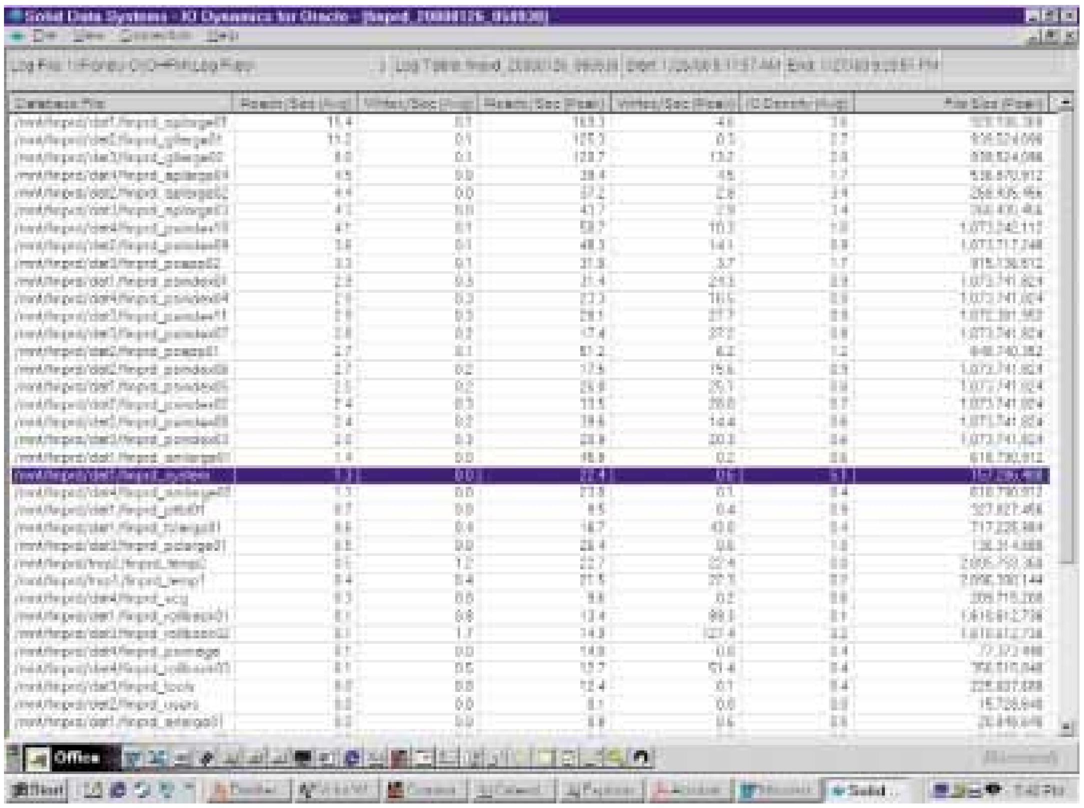
I/O Dynamics for Oracle

For an Oracle application, DBAs and system administrators need tools that help them identify the most I/O-intensive files or tables. A good tool for this purpose is available for free download at http://www.soliddata.com/products/iodreg.html.

Solid Data's I/O Dynamics™ for Oracle is a tool for monitoring the file activity of an Oracle Database. It can help identify files with high levels of I/O activity, which may be having a significant impact on overall system performance. It can monitor and analyze one or more running Oracle instances. It is a self-contained client application that runs on a Windows client, so none of the processing work is offloaded to the Oracle server, and it does not pose a risk to the integrity or performance of the Oracle database that it is monitoring.

Figure 7 shows the results of hot file analysis on the Peoplesoft application we considered in Figure 6. The display combines I/O rate data for each file (reads per second and writes per second) with file size information for each file. It also presents a computed ratio called I/O density, which reflects the percentage of overall I/O activity represented by the file, divided by its percentage of the total storage consumed by the application.

These statistics allow the user to identify files that represent a large percentage of the I/O on a small amount of total disk space, and to target those files for relocation to a solid-state file cache.



Solid Data Systems - IO Dynamics for Oracle - [fespd_20000126_054936]

File View Connection Help

Log File: \\Fonts\ O\O=R\Log Files | Log Table: fespd_20000126_095508 | Start: 1/26/00 8:17:57 AM | End: 1/27/00 9:09:51 PM

Database File	Reads/Sec (Avg)	Writes/Sec (Avg)	Reads/Sec (Peak)	Writes/Sec (Peak)	IO Density (Avg)	File Size (Peak)
/mnt/fespd/dat7/fespd_aplarge07	11.4	0.1	163.3	4.6	7.6	523,730,368
/mnt/fespd/dat2/fespd_gllarge01	11.2	0.1	125.3	0.3	2.7	838,524,096
/mnt/fespd/dat3/fespd_gllarge02	8.0	0.1	123.7	13.2	2.0	838,524,096
/mnt/fespd/dat4/fespd_aplarge04	4.5	0.0	28.4	4.5	1.7	536,870,912
/mnt/fespd/dat2/fespd_aplarge02	4.4	0.0	37.2	2.8	3.4	268,435,456
/mnt/fespd/dat3/fespd_aplarge03	4.1	0.0	41.7	2.9	3.4	268,435,456
/mnt/fespd/dat4/fespd_poindex10	4.1	0.1	53.7	10.3	1.0	1,073,242,112
/mnt/fespd/dat2/fespd_poindex09	3.6	0.1	48.3	14.1	0.9	1,073,717,248
/mnt/fespd/dat3/fespd_poapp02	3.3	0.1	31.8	3.7	1.7	815,136,512
/mnt/fespd/dat1/fespd_poindex01	2.3	0.3	31.4	21.3	0.9	1,073,741,824
/mnt/fespd/dat4/fespd_poindex04	2.0	0.3	23.3	16.5	0.9	1,073,741,824
/mnt/fespd/dat3/fespd_poindex11	2.0	0.3	28.1	27.7	0.8	1,072,381,952
/mnt/fespd/dat3/fespd_poindex07	2.0	0.2	17.4	27.2	0.8	1,073,741,824
/mnt/fespd/dat2/fespd_poapp01	2.7	0.1	51.2	6.2	1.2	848,740,352
/mnt/fespd/dat2/fespd_poindex08	2.7	0.2	17.5	15.6	0.9	1,073,741,824
/mnt/fespd/dat1/fespd_poindex05	2.5	0.2	25.8	25.1	0.9	1,073,741,824
/mnt/fespd/dat7/fespd_poindex02	2.4	0.3	13.5	28.0	0.7	1,073,741,824
/mnt/fespd/dat5/fespd_poindex06	2.4	0.2	19.6	14.4	0.6	1,073,741,824
/mnt/fespd/dat3/fespd_poindex03	2.0	0.3	23.9	20.3	0.6	1,073,741,824
/mnt/fespd/dat1/fespd_arlarge01	1.4	0.0	45.9	0.2	0.6	618,790,912
/mnt/fespd/dat1/fespd_system	1.3	0.0	22.4	0.6	5.1	157,286,400
/mnt/fespd/dat4/fespd_arlarge02	1.3	0.0	23.8	0.1	0.4	618,790,912
/mnt/fespd/dat1/fespd_ptfd01	0.7	0.0	8.5	0.4	0.9	327,827,456
/mnt/fespd/dat1/fespd_fclarge01	0.6	0.4	18.7	43.0	0.4	717,225,984
/mnt/fespd/dat2/fespd_polarge01	0.5	0.0	28.4	0.6	1.0	136,314,880
/mnt/fespd/tmp2/fespd_temp2	0.5	1.2	22.7	22.4	0.0	2,005,753,368
/mnt/fespd/tmp1/fespd_temp1	0.4	0.4	21.5	27.5	0.7	2,098,200,144
/mnt/fespd/dat4/fespd_ecg	0.3	0.0	9.8	0.2	0.8	209,715,200
/mnt/fespd/dat1/fespd_rollback01	0.1	0.8	13.4	99.3	0.1	1,616,812,736
/mnt/fespd/dat3/fespd_rollback02	0.1	1.7	14.3	127.4	2.2	1,616,812,736
/mnt/fespd/dat4/fespd_psmege	0.1	0.0	14.9	0.0	0.4	77,373,440
/mnt/fespd/dat4/fespd_rollback03	0.1	0.5	12.7	51.4	0.4	356,515,840
/mnt/fespd/dat3/fespd_tools	0.0	0.0	12.4	0.1	0.4	225,837,888
/mnt/fespd/dat2/fespd_users	0.0	0.0	8.1	0.0	0.0	15,728,640
/mnt/fespd/dat1/fespd_arlarge03	0.0	0.0	9.8	0.6	0.5	20,846,[illegible]

Figure 7. Hot File Identification with I/O Dynamics for Oracle

Architectural Adoption

As database administrators, application developers and capacity planners become more familiar with the benefits of solid-state file caching in specific applications, they start to adopt the technology more broadly. Figure 8 illustrates one attractive aspect of an architecture that combines cached disk arrays and solid-state file cache: it becomes possible to add performance and capacity as independently scalable features of the infrastructure.

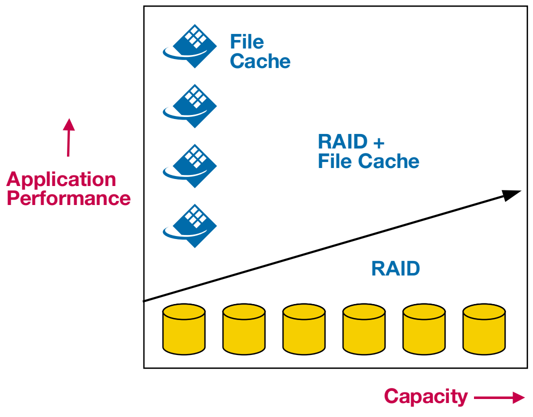


Figure 8. Independent Scalability of Application Performance and Storage Capacity

Looking Ahead

In the future, Storage Area Networks (SANs) will be widely deployed and supported by sophisticated storage management tools—such as virtual storage architectures and policy-based storage management consoles (see Figure 9).

As a shared facility on the SAN, solid-state storage will be easy to deploy and easy to manage. As this development occurs, file-cache appliances will prove increasingly attractive and cost-effective for architectural deployment in a wide range of transaction-intensive applications.

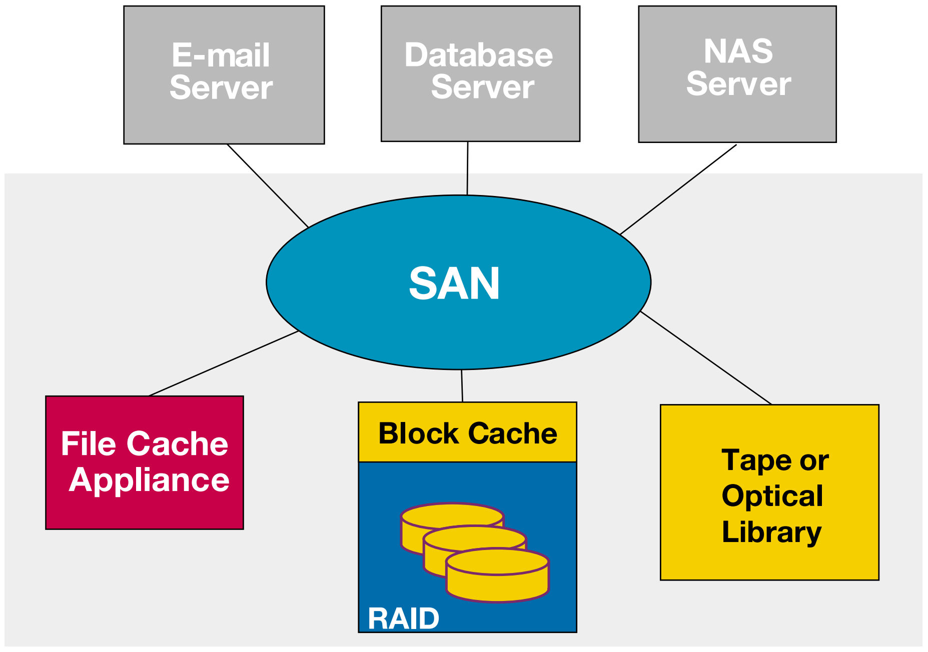


Figure 9. File Cache as
Shared Storage on the SAN

Conclusion

When scaling mission-critical applications, enterprises can avoid costly mistakes by understanding the data-access patterns inherent in their applications, and using appropriate software tools to determine whether mechanical disk I/O is limiting performance and scalability. In transaction intensive applications, when the servers are wasting a lot of their time waiting for reads and writes to a few hot files, enterprises should consider solid-state file-caching. This architectural innovation multiplies server performance and enables enterprises to sustain rapid growth with scalable, cost-effective infrastructure.