CS546 "Parallel and Distributed Processing" Fall 2019
– Homework 5 (I/O lectures)

1. Provide definitions for: a) file, b) file metadata, c) file operations, and d) file attributes.

a) **file:** Named collection of logically related data A file is an object on a computer that stores data, information, settings, or commands used with a computer program.

b) **file metadata:** Maintained by the file system Metadata is information stored in almost any type of file. It can include your name, your company or organization's name, the name of your computer, the name of the network server or drive where you saved the file, personalized comments and the names and times of previous document authors, revisions, or versions.

c) **file operations:** Create, delete, open, close, read, write, append, get/set attributes.

d) **file attributes:** varies from OS to OS – Name, type, location, size, protection info, password, owner, creator, time and date of creation, last modification, access.

2. What is a file system and what are its main components?

- Provides a logical view of data and storage functions. In computing, a file system or filesystem (often abbreviated to fs), controls how data is stored and retrieved
- User-friendly interface, Provides facility to create, modify, organize, and delete files .
- Provides sharing among users in a controlled manner, Provides protection.
- **Components**: directory, authorization, file service and system service
    - Authorization service: between file and directory services.
    - Directory service: used to keep track of the location of all resources in the system.
    - File service provides a transparent way of accessing any file in the system in the same way.
    - System service: file system's interface to hardware

3. Are the open and close system calls strictly necessary? How would a system work without them?

- open is a system call that is used to open a new file and obtain its file descriptor.
- A close system call is a system call used to close a file descriptor by the kernel. For most file systems, a program terminates access to a file in a filesystem using the close system call.
- Open and Close are not strictly necessary but without them working with files would be extremely complex.
- We can have fixed size files, and filename can be exactly the same as the actual physical address, then we can directly seek to that position and access a file without Open syscall. similarly, without Close(), the programmer has to remove any locks that he has on file or System might end up in a deadlock.

4. Provide an overview of PFS. Briefly discuss its design goals, its strengths and weaknesses.

- Overview: A parallel file system is a software component designed to store data across multiple networked servers and to facilitate high-performance access through simultaneous, coordinated input/output operations (IOPS) between clients and storage nodes.
- Design Goals: A parallel file system breaks up a data set and distributes, or stripes, the blocks to multiple storage drives, which can be located in local and/or remote servers. Users do not need to know the physical location of the data blocks to retrieve a file. The system uses a global namespace to facilitate data access
- Strengths and weakness:
    - Parallel file systems often use a metadata server to store information about the data, such as the file name, location and owner.
    - A parallel file system reads and writes data to distributed storage devices using multiple I/O paths concurrently, as part of one or more processes of a computer program. The coordinated use of multiple I/O paths can provide a significant performance benefit, especially when streaming workloads that involve a large number of clients.
    - Capacity and bandwidth can be scaled to accommodate enormous quantities of data. Storage features may include high availability, mirroring, replication and snapshots.

5. Compare the file-per-process and shared-file access patterns. Pros and cons.

File-per-process (N to N): Each application task creates a separate file and writes to only that file.

- Avoids lock contention on file systems to use locks to maintain POSIX consistency.
- Applications running today creates as many as 100000 tasks.
- Impossible to restart application with different number of tasks.

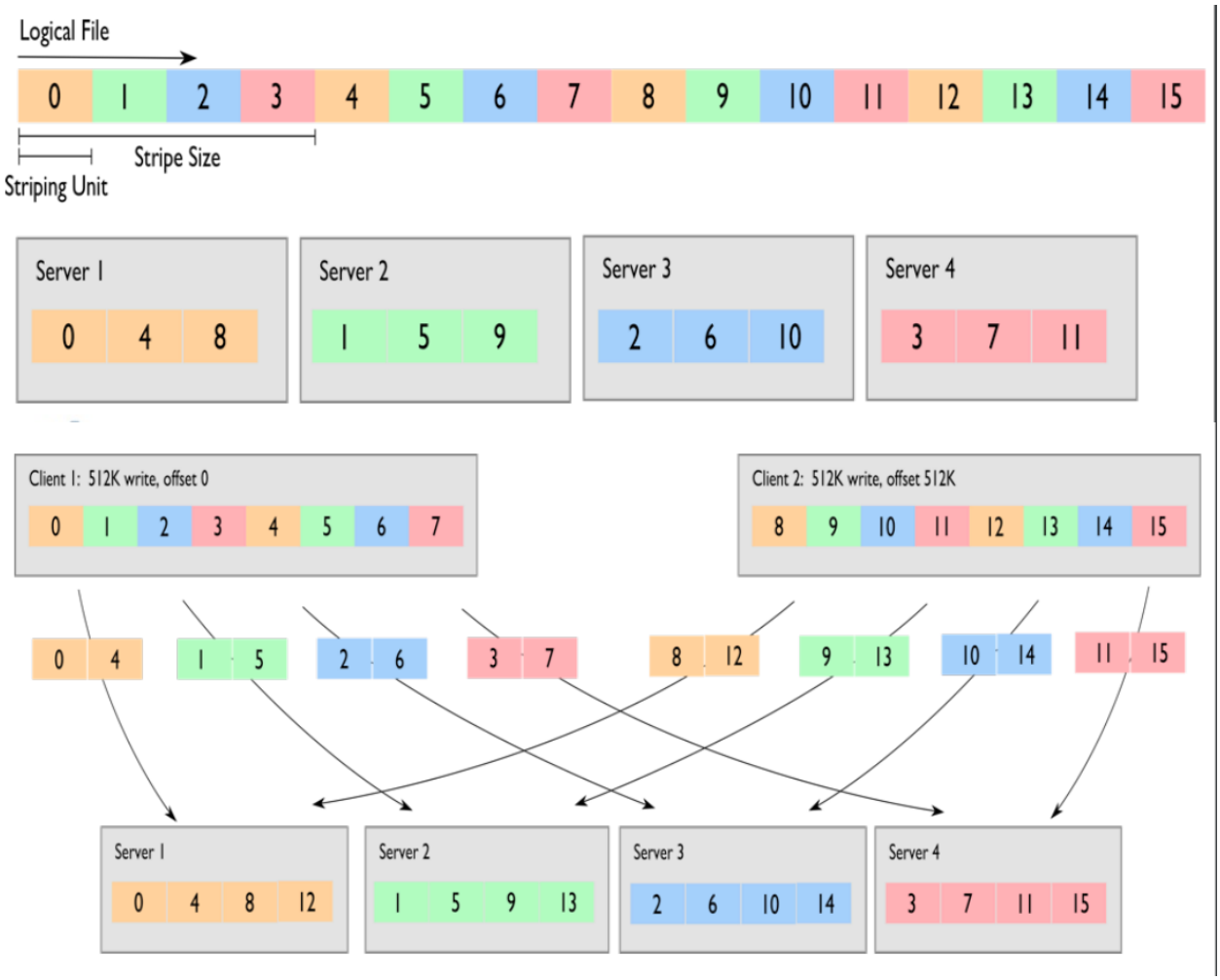
Shared-file (N-to-1): A single file is created, and all application on tasks write to that file (usually to completely disjoint regions)

- Increases usability: only one file to keep off by application.
- Can create lock contention and hinder performance on some system .

6. Describe how data distribution works in PFS.

- Logically a file is an extendable sequence of bytes that can be referenced by offset into the sequence.
- Meta data associated with the file specifies a mapping of this sequence of bytes into a set of objects on PFS servers.

- Extents in the byte sequence are mapped into objects on PFS server. This mapping is usually determined at file creation time and is often a round-robin distribution of a fixed size over the allocated objects.
- Space is allocated on demand, so unwritten “holes” in the logical file do not consume disk space.
- A static mapping from logical file to objects allows clients to easily calculate servers to contact for specific regions, eliminating need to interact with a metadata server on each I/O operation.
- Round-Round is a reasonable default solution .
- It works consistently for a variety of workloads.
- Works well on most systems.
- Clients perform read and write operations of file at various regions. It usually depends on application workloads and number of tasks. This is the way data distribution works in Parallel file system.



7. Locking in PFS. Discuss and list some of the challenges.

- File are broken up into lock units. Clients obtain locks on units that they will access before I/O occurs.
- Enables caching on clients as well (as long as clients has a lock, it knows its cache data is valid)
- Client can optimize small I/O with readahead. Locks are reclaimed from clients when others desire access.

- Locks are delegated and revokes through distributed lock managers.
- Challenges are

    - Implementation burden: DLMs add complexity to file system
      :Chances of DLM node failure

    - Locks are expensive: Round-trip latencies between clients and DLM
      :Problem of client failure

8. What is Layered Performance Matching (LPM) method? What is the advantage of the LPM method? Why LPM can improve memory performance (in terms of memory stall time) by more than one hundred times?

- Layered Performance Matching is a method to match the data access cycles at each layer of the memory hierarchy with a view of improving the total memory performance.
- **The advantages of LPM are:** It considers both the data access concurrency and locality. The memory concurrency can mask the data access delay.

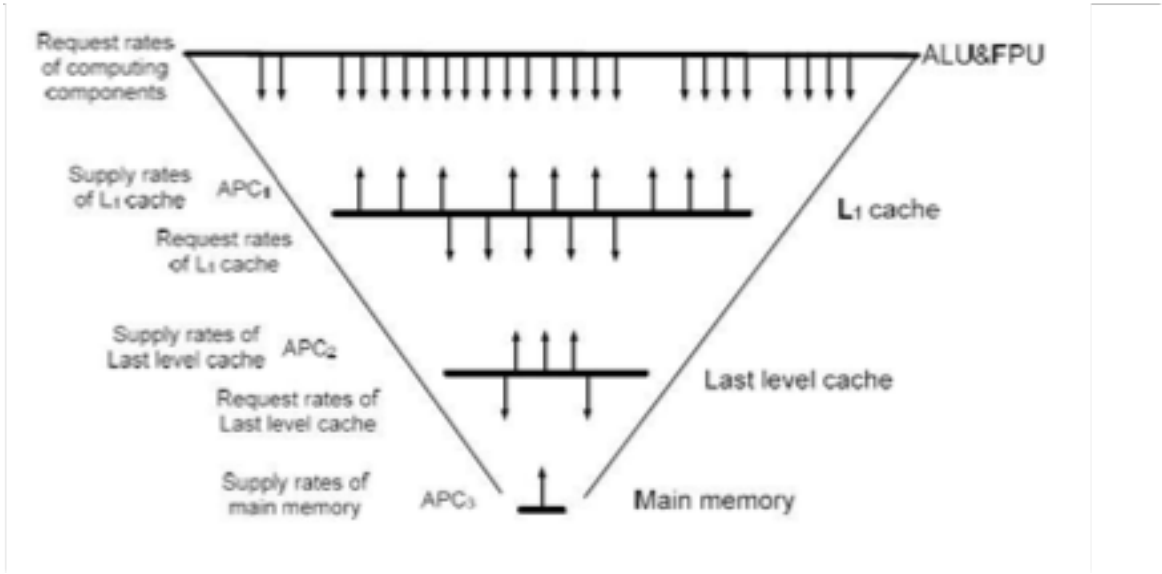


Figure: The request-reply LPM model

Impact of LPM on Memory Stall Time:

- LPM is a concurrency driven matching method in a memory hierarchy. It focuses on data access concurrency to optimize the matching between request and supply at each memory layer.

- By optimizing the performance at each memory layer, it ensures effective data access between the layers.

- This optimality at every memory layer turns out to reduce the memory stall time and thereby improves the overall memory performance.

- The resulting performance will be much greater than the actual memory stall time and hence the LPM can improve memory performance (in terms of memory stall time) by more than one hundred times.

9. What is Pace-Matching Data Transfer? Can we achieve Pace-Matching Data Transfer?

Based on this design, a memory-computing hierarchy is used to mask the performance gap between the computing and the actual data transfer.

- **C-AMAT** is used to compute the data transfer request/supply ratio at each layer of the memory hierarchy.
- In addition to this, a global control algorithm called **Layered Performance Matching (LPM)**, which matches the data transfer at each memory layer and thereby match the overall performance between the CPU and the underlying memory system.
- This pace-matching optimization is contrary to the conventional locality-based system optimization and can minimize the memory-wall effects.
- **We can achieve Pace-Matching Data Transfer** under certain assumptions:

1. The applications should have data concurrency and the system should have sufficient hardware for supporting the data concurrency.
2. The architecture needs to be elastic to accommodate different data access patterns for any given application.

- The **Sluice-Gate Theory** is a framework which solves the memory wall problem considering the above assumptions.

- Therefore, it can be concluded that pace-matching data transfer can be achieved by eliminating the memory wall problem both theoretically and practically by engineering solutions.

10. Assume: CycleCPU = 2 ns, Cyclemem = 8 ns, fmem = 20%, IPCexe = 2.5, APC = 1; please calculate the LPM Ratio (LPMR).

LPMR = $IPCexe \times fMem / APC$

$= 2.5 \times 20 / 1 \times 100$

$= 1/2$ LPMR = 0.5

• In 8ns, there is 1 Memory Cycle, and the data supply rate is: APC x $N\_Cycle_{Mem} = 1$ • In 8ns, there is 4 CPU Cycle, and the data request rate is: IPC_{exe} x N_Cycle_{CPU} x $f_{Mem} = 2$
• So, the data supply rate does not match the data request rate. • We can decrease the CPU frequency (increase CPU cycle) to adjust the data request rate.

- For example, increase the CPU cycle from 2ns to 4ns, the new N_Cycle_{CPU} would be 2, so:

$IPC_{exe} \times N\_Cycle_{CPU} \times f_{Mem} = 1$

• Now, the data request rate matches the data supply rate.

11. Give two non-locality-based methods which can reduce C-AMAT. Please explain why your methods works.

C-AMAT can be improved by tweaking components in hardware, compilers and applications. Below is the summary of techniques that improve C-AMAT stalls which will inturn help improving C-AMAT.

Classes	Items	$C\text{-}AMAT_{stall}$
Hardware techniques	Pipelined cache access	⊕
	Non-blocking caches	⊕
	Multi-banked caches	⊕
	Large IW & ROB, Runahead	⊕
	SMT	⊕
Compiler techniques	Loop Interchange	⊕
	Matrices blocking	⊕
	Data and control dependency related optimization	⊕
Application techniques	Copy data into local scalar variables and operate on local copies	⊕
	Vectorize the code	⊕
	Split structs into hot and cold parts, where the hot part has a pointer to the cold part	⊕