Erasure coding vs replication

These two strategies each have their advantages and disadvantages. I'll cover each aspect individually. In general, many of these aspects are affected more by the implementation decisions than core concept, but I'll attempt a general assessment also.

As we've learned in class, **network IO** is often the most expensive resource. The replication implementation is obviously far more expensive in terms of total bytes transferred, even with a replication factor of 2 (obviously the number of parity shards also affects this computation). However, in this implementation, the number of messages sent by the client is far greater for erasure coding, since each file is chunked *then* broken into shards. This could be easily mended by only chunking for replication, while sending a constant n (k+m) messages for sharding (of course then messages are variable size). Still, for the overall fewer bytes transferred, the clear winner here is **Erasure Coding**.

Similar to network IO, **disk IO** is an expensive, often limiting factor in distributed systems. In terms of total storage footprint, there is a clear winner. Obviously, replication requires distributing the entire file R times over (where R is the replication factor). Erasure coding is also configurable to produce a larger/smaller footprint, but is in general much more space efficient (even optimal, apparently). Again, the number of reads and writes is implementation-specific. If, however, we are doing slice-level recovery for replication (as in this project), we are doing many more random read/writes during recovery and validation than erasure coding. So, the clear winner is **Erasure Coding.**

Next, **CPU time** is a crucial resource, and the winner is again clear. The replication strategy, even as implemented, is almost all copying and hashing. Both of these operations are very fast on modern CPUs. Erasure coding, however, is much more complex. The best time complexity out there is apparently polynomial, obviously much worse than replication. Winner: **Replication**.

These remaining aspects are purely implementation-specific. In terms of **fault-tolerance**, our implementation requires the client to recover lost shards in erasure coding. However, the **replication strategy** is recovered by the controller/chunk servers as soon as a fault is detected. In terms of **programming complexity**, my opinion favors **erasure coding**. We didn't write the library, and no hashing and slicing and exchanging of arbitrary file fragments is necessary.

So, the clear winner here is **erasure coding**. This is dependent on implementation decisions, what the system might be expected to accomplish, and what computational resources are available.