The Google File System (GFS), as described by Ghemwat, Gobioff, and Leung in 2003, provided the architecture for scalable, fault-tolerant data management within the context of Google. These architectural choices, however, resulted in sub-optimal performance in regard to data trustworthiness (security) and simplicity. Additionally, the application-specific nature of GFS limits the general scalability of the system outside of the specific design considerations within the context of Google.

SYSTEM DESIGN:

The authors enumerate their specific considerations as: (1) commodity components with high expectation of failure, (2) a system optimized to handle relatively large files, particularly multi-GB files, (3) most writes to the system are concurrent append operations, rather than internally modifying the extant files, and (4) a high rate of sustained bandwidth (Section 1, 2.1). Utilizing these considerations, this paper analyzes the success of GFS's design in achieving a fault-tolerant, scalable system while also considering the faults of the system with regards to data-trustworthiness and simplicity.

Data-trustworthiness: In designing the GFS system, the authors made the conscious decision not prioritize data trustworthiness by allowing applications to access 'stale', or not up-to-date, data. In particular, although the system does inform applications of the chunk-server version number, the designers of GFS encouraged user applications to cache chunkserver information, allowing stale data accesses (Section 2.7.2, 4.5). Although possibly troubling in a general context, the system designers accepted this eventuality in their design based on the requirements at Google.

Simplicity: The designers of GFS implemented significantly more complicated storage mechanisms to account for performance optimizations, as evidenced by comparing the seven steps required for GFS's lease-to-write mechanism to the two steps required by the Sprite Log File System (Section 3.1) (Rosenblum and Ousterhout, Section 3).[1] While certainly optimized for completely different applications, the performance goals of the

[1] It can certainly be argued that this comparison ignores the complexity of segment cleaning, however GFS's architecture requires a similar level of complexity in garbage collection (Section 4.4).

designers of GFS mean that the system implementation required a complicated system when optimized over a distributed system with many concurrent reads and appends.

Fault-tolerance: The designers of GFS provided a high-degree of fault-tolerance across the system utilizing techniques including load-balancing, check-summing, and many other features. This consideration is particularly evident in the replicated structure of chunkservers. Specifically, the master server for GFS is strictly replicated across multiple machines, and, in turn, the master ensures that all chunkservers are replicated after some duration of time (Section 5). The system-wide mechanism of replication ensures that any data losses due to component failures, an assumption made by the designers, are minimized.

Scalability: Modularity allows GFS to easily expand to account for increasing amounts of data and users. The paper states that currently the system accounts for approximately 300 TB of information (Section 1); however, the system is designed such that adding more chunkservers can be accomplished without significantly modifying the master server (Section 2). Further, the decentralized method of data access that primarily involves chunkserver-application interaction alleviates significant bottlenecks at the master (Section 2). The combination of extensibility as well as performance across increasing amounts of users and data means that the system is entirely scalable within the Google context.

ANALYSIS

Although certainly fault-tolerant and scalable within the context of Google, as discussed above, the specificity of the design of GFS minimizes the generalizability of the system as a whole. To further understand this distinction, this paper will analyze GFS using two use cases: (1) a Google application and an (2) individual users.

Google Application: Consider some instance of a Map-Reduce algorithm that is constructing a map of users to buying preferences (i.e. for ads). GFS is certainly a viable choice for use in this application as it will (1) generate a large (GBs) amount of static,

sequential data and (2) primarily requires reads from the data in the future (i.e. point a search query to the correct position). GFS is certainly an excellent system for this application, as the query can generate and access a large amount of data in a distributed fashion.

Individual Application: Individual users, utilizing GFS to write, for example, a combined paper critique for a systems design course, would not be able to effectively utilize GFS. For example, individual modifications and files composed at a small scale are particularly troubling for GFS. Further, the loose restriction on data integrity would mean that discrete machines accessing different versions of the chunkserver might threaten the integrity of the data.

CONCLUSION

The designers or GFS, describe a system that is scalable and fault-tolerant within the general considerations of Google's data management needs, yet lacks overall generalizability to dissimilar data management tasks. The Google File System imposes task-specificity within the system to optimize performance in exchange for lowered usefulness in dissimilar data-management tasks.

Works Cited

1. Ghemwat, Sanjay, Howard Gobioff, and Shun-Tak Leung (Google). “The Google File System.” Symposium on Operating Systems Principles (SOSP) ’03, Association of Computing Machinery. Published 2003.
2. Rosenblum, Mendel, and John K Ousterhout. “The Design and Implementation of a Log-Structured File System.” Symposium on Operating System Principles (SOSP) ’92, Association of Computing Machinery. Published 1992.

Note on citation: all parenthetical citations attributed to Rosenblum and Ousterhout are written as: (Rosenblum and Ousterhout, Section #). All other citations refer to the paper by Sanjay, Gobioff, and Leung.