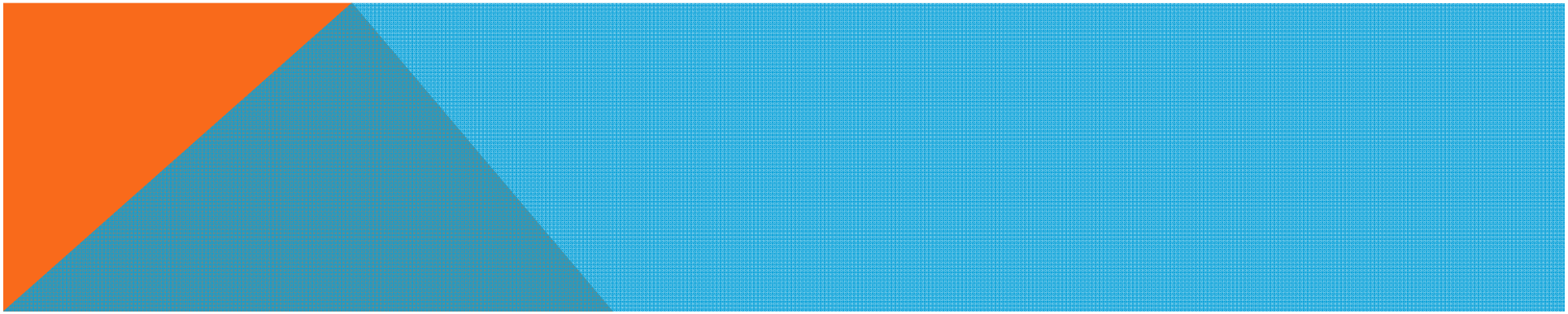
THE GOOGLE FILE SYSTEM

BY: CHRIS DELLADONNA

MAIN IDEA:

- The Google file system is a scalable , distributed file system for large distributed data-intensive applications
  - runs on inexpensive commodity hardware
  - provides fault tolerance
  - delivers high aggregate performance to all
- GFS file system also supports all of their storage needs
  - Appending data is the focus of performance optimization
  - Block size is 64MB, since they are mostly storing small files
  - Can still effectively store multi-GB files
- The GFS allows users to work on files concurrently
  - Users can read, write, and edit at the same time as other users
  - They have prioritized high-sustained bandwidth for processing needs, without emphasis on connection time because of application demand.



HOW IT IS IMPLEMENTED:

The GFS has been implemented with a lot of specific topics in mind.

- It is scaled over a lot of inexpensive commodity hardware
  - Separated into a single-Master and chunk servers over a simplified interface
    - Files are stored Hierarchically
  - Expect various failures and holdups to occur
    - Focus had to be put into error and fault detection
- GFS provides a lot of space and is extremely scalable
  - Adding space to chunk servers is very cheap
  - Expected the master could be a potential bottleneck in the system as a result of the amount of chunk pointers that it stores.
    - Each pointer is roughly 64 bits, meaning that the space is accounted for
    - Adding space to the master is also cheap and easy
- Constant mutations from multiple connections are kept consistent
  - logging mutations to an operation log , which is stored on the master's local disk and remotely for reliability
- The system also has the ability to consistently scan through its entire state in the background.
  - used for chunk garbage collection
  - re-replication in the presence of chunk server failure,
  - chunk migration to balance load and disk space
- Designed to reduce network overhead

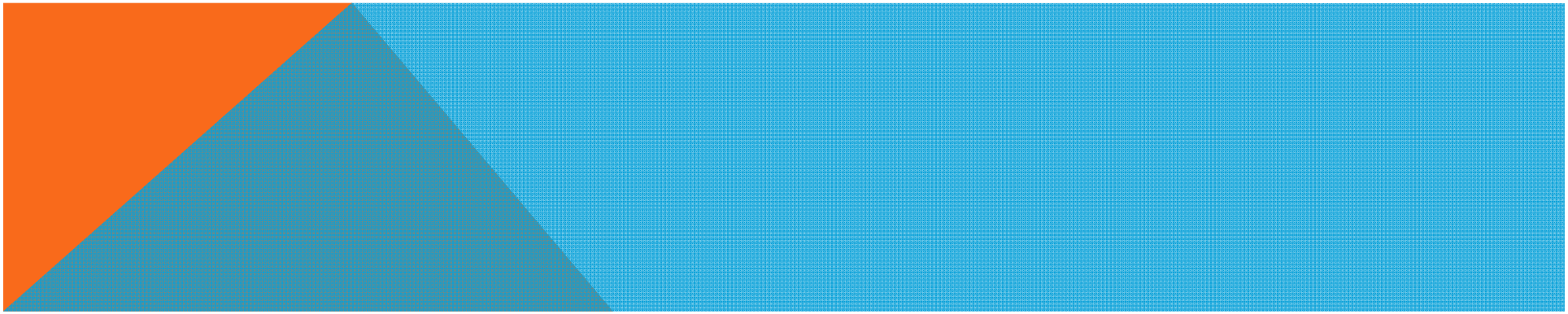
GFS polls chunk servers for that information at startup

Master keeps itself up to date by controlling chunk placement and monitoring chunk server status with regular heartbeat messages.

MY ANALYSIS

In my opinion, the GFS engineering team has seemed to account for everything that they could experience during the lifespan of the system by:

- Having fantastic idea how to deal with issues that will more than likely occur resulting from running the system on commodity hardware
- Implementing heartbeat monitors constantly check for failures and interruptions of chunk servers
- Accounting for the amount of space that they should expect to need in the future.
- Realizing the obvious bottlenecks in designing the system the way that they did
  - adding space to the chunk servers is very easy for the degree of service that it provides
- Achieving high reliability by keeping a method to recover information even in catastrophic situations.
  - with the snapshot feature, they always have a checkpoint to look back to
    - if there was an error in creating the checkpoint, that checkpoint will be skipped when the system restarts and it will go back the checkpoint beforehand.



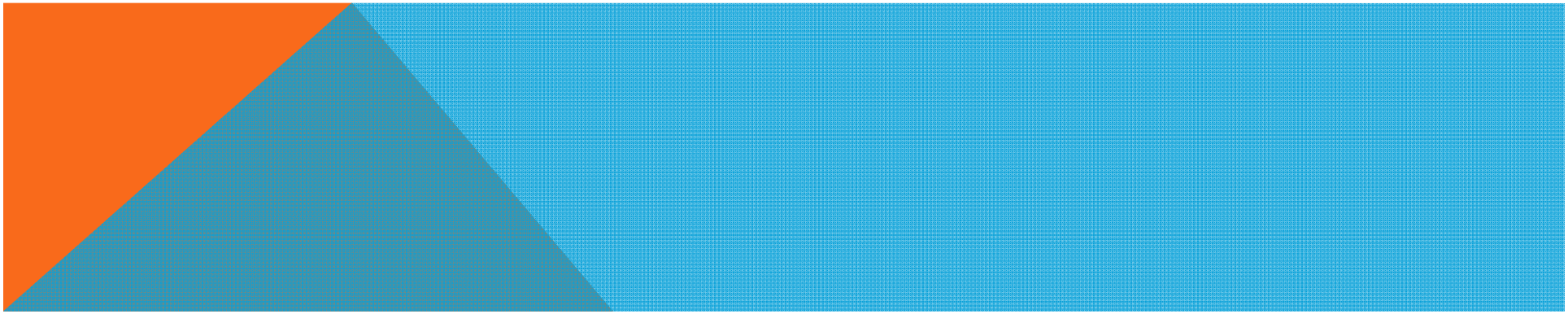
ADVANTAGES/DISADVANTAGES:

Advantages

- The file system is very realizable because of the implemented status scanning of the servers involved
- Manages space very efficiently
- Handles traffic based on their needs and their analysis of how most people use their system
- Very fast; the GFS puts the processing load on the host computer and on the chunk servers; very minimal on the masters
- Each chunk server also allows concurrent appends to occur making it very useful for common tasks shared by a group of people.

Disadvantages:

- Quantity and quality of the components virtually guarantee that some are not functional at any given time
- Upon many people connecting to the same chunk, the chunk can become a hotspot, which can lead to overloading the server with traffic
    - (NOTE: They have fixed this problem for now by storing such executables that affect several chunks at the same time with a higher replication factor and stagger application startup times. However, this is not a long-term solution).



REAL WORLD CASES:

Real world cases of this technology, I would say, mostly consists of the use from everyday people.

Common folk:

- The average person can effectively access, share, and modify files easily and without high demand on their personal computers.
- They can also store application data for business needs and have the ability to access that information from any host computer that has internet connection.

Students:

- It allows the common student to work on any amounts of projects/schedules/homework/notes on the cloud without having to worry about saving your work or loosing what you have done.
- It also makes group work very simple because you can concurrently append documents without getting in the way of other people
  - That file does not have a very limited size and it can also be accessed by any person that has efficient permission to access it.