Topic 2: Google Infrastructure and GFS

- What are the design principles behind the Google Infrastructure?
- Go into depth with GFS
- What kind of infrastructure does GFS target? What are the key assumptions and design goals behind it?
- Explain the architecture, consistency model, replication, fault tolerance
- What are its advantages and disadvantages?

Material: slides on Google infrastructure, GFS paper, “Web-search for a planet” paper

Design Principles

Energy efficiency and price-performance ratio. (Web-search for a planet, page 1)

GFS

GFS targets an infrastructure composed of many commodity hardware components.

GFS was created with the following assumptions:

- Failures are the norm
- Files are typically very large (multi-gigabyte)
- Large streaming reads (sequential) and small random reads (often buffered, also sequential)
- Write operations are typically append, they are typically large and sequential
- Support concurrent appending to a file with well-defined semantics
- High sustained bandwidth is more important than latency

Architecture

The high-level architecture is composed of three actors: One master, many chunkservers, many clients. All run Linux.

The master acts as a DNS server for clients, i.e. it takes a request and returns an address to a chunkserver. Chunks are 64MB each, and have unique 64-bit identifiers: *chunk handles*.

Consistenty Model

The paper describes the model as a "relaxed consistency model". Garbage is collected lazily. It guarantees:

- Atomic file namespace mutations, such as file creation.
- A file region is:
  - *Consistent* if all clients see the same data from all replicas.
  - *Defined* if consistent and clients see their writes in their entirety.
- Records are appended atomically at least once somewhere
- Applications can handle inconsistent regions

Replication

Replicas are spread across different racks. This makes them very resistant to faults, but increases the time it takes to write. The user defines a replication goal, which is typically 3.

Replicas are created for three reasons:

1. **Creation**: Initially, the replicas are empty
2. **Re-replication**: When replicas are lost, or replication goal redefined.
3. **Rebalancing**: Periodical rebalancing

Considerations: Resource utilization, recent creates, and separate racks.

Fault Tolerance

1. Constant monitoring
2. Replicating crucial data
3. Fast and automatic recovery

Conclusion

Only appropriate for very specific stacks. Does not support small files very well. High latency, especially with writes.