This article gives a detailed introduction to the Bigtable storage system, it is designed for managing the structured data to satisfy both bulk processing and real-time data serving requirements. Lots of Google’s online service like Google web indexing, Google Earth, and Google Finance are using this system. Compared to the parallel databases and main-memory databases, Bigtable provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage instead of providing a full relational data model.

Then the paper introduced the model of Bigtable. Basically, Bigtable’s data model is a sorted map with sparse, distributed, persistent multi-dimensional features. The map is indexed by the row key, column key and timestamps; each value in the map is a uninterpreted array of bytes. This structure is interesting since it gives me the reason that why the storage system is called Bigtable.

To implement Bigtable, there is a library that's linked into every client, a master server and many tablet servers. It's built on the top of GFS and operates in a shared pool of machines that run a wide variety of other distributed applications. Bigtable’s data are storing in the Google SSTable, in this manner, Bigtable can provide high performance lookup and can be mapped into memory to omit extra disk lookup. Updates of a tablet is first committed to a commit log that store redo records. Most recent committed updates are stored in *memtable*, older ones are stored in a sequence of SSTables. When the memtable grows into a certain size, it will be compacted into SSTable. By using this technique, Bigtable can firstly shrink the memory usage of the tablet server and secondly reduce the amount of data that has to be read from the commit log during recovery if the server dies. A major compaction is scheduled regularly to produce SSTable that contains no deletion information or deleted data.

In the end, the paper talked about the performance and refinements. To improve the performance, bigtable uses two-level caching and bloom filters. Scan Cache caches key-value pairs returned by SSTable interface, and Block Cache caches results returned from GFS. Bloom filters can reduce the group of servers that a read operation need to contact thus reduce the number of disk accesses.

The good part of this paper is that the paper provides many real-world implementation examples with detailed figures and table to help the readers understand the working progress and performance. There are also some weak points. For example, in the refinements section, Bigtable uses Bloom ﬁlter to reduce the number of disk access. However, what if the Bloom ﬁlter fails to reduce the access? Are there any backup measures? Also, the paper only changes the server number to show Bigtable performance under different conditions, it can be more persuasive if there are some comparison across different storage systems.